

Entwicklerhandbuch

AWS SDK for Rust



AWS SDK for Rust: Entwicklerhandbuch

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

Die Marken und Handelsmarken von Amazon dürfen nicht in einer Weise in Verbindung mit nicht von Amazon stammenden Produkten oder Services verwendet werden, die geeignet ist, Kunden irrezuführen oder Amazon in irgendeiner Weise herabzusetzen oder zu diskreditieren. Alle anderen Handelsmarken, die nicht Eigentum von Amazon sind, gehören den jeweiligen Besitzern, die möglicherweise zu Amazon gehören oder nicht, mit Amazon verbunden sind oder von Amazon gesponsert werden.

Table of Contents

Was ist der AWS SDK for Rust?	1
Erste Schritte mit dem SDK	1
Wartung und Support für SDK-Hauptversionen	1
Weitere Ressourcen	2
Erste Schritte	3
SDK-Authentifizierung mit AWS	3
Starten Sie eine AWS Access-Portal-Sitzung	4
Weitere Authentifizierungsinformationen	5
Hallo Tutorial	6
Voraussetzungen	6
Erstellen Sie Ihre erste SDK-App	6
Grundlagen	8
Voraussetzungen	6
Grundlagen von Rust	9
AWS SDK for Rust Grundlagen erstellen	10
Projektkonfiguration für die Arbeit mit AWS-Services	11
Tokio-Laufzeit	11
Konfiguration	13
Erstellen Sie einen Service-Client	13
Konfigurieren Sie einen Client aus der Umgebung	14
Verwenden Sie das Builder-Muster für dienstspezifische Einstellungen	15
Erweiterte explizite Client-Konfiguration	16
Verhaltensversionen	17
Stellen Sie die Verhaltensversion ein <code>Cargo.toml</code>	18
Legen Sie die Verhaltensversion im Code fest	18
Anbieter von Anmeldeinformationen	18
Die Kette der Anbieter von Anmeldeinformationen	19
Anbieter für explizite Anmeldeinformationen	22
Zwischenspeichern von Identitäten	22
AWS-Region Auswahl	22
AWS-Region Anbieterkette	23
Konfigurationsoptionen	23
Client-Endpunkte	24
Benutzerdefinierte Konfiguration	25

Beispiele	28
HTTP	30
Hyper 1.x	30
Abfangjäger	32
Registrierung des Interceptors	33
Beobachtbarkeit	34
Protokollierung	35
Überschreiben Sie die Betriebskonfiguration	39
Wiederholversuche	40
Standardkonfiguration für Wiederholungsversuche	40
Maximale Anzahl an Versuchen	41
Verzögerungen und Rückschläge	41
Adaptiver Wiederholungsmodus	42
Timeouts	43
API-Timeouts	43
Der Stream-Schutz ist ins Stocken geraten	45
Verwenden Sie das SDK	47
Anfragen stellen	47
Bewährte Methoden	48
Verwenden Sie SDK-Clients nach Möglichkeit wieder	48
API-Timeouts konfigurieren	49
Nebenläufigkeit	49
Bedingungen	49
Ein einfaches Beispiel	50
Eigentum und Wandelbarkeit	51
Weitere Begriffe!	52
Wir haben unser Beispiel umgeschrieben, um es effizienter zu machen (Single-Thread-Parallelität)	53
Wir haben unser Beispiel umgeschrieben, um es effizienter zu machen (Parallelität mit mehreren Threads)	55
Debuggen von Apps mit mehreren Threads	57
Fehlerbehandlung	57
Dienstfehler	57
Fehler-Metadaten	58
Detaillierter Fehler beim Drucken mit <code>DisplayErrorContext</code>	59
Lambda-Funktionen erstellen	61

Paginierung	62
Vorsigniert erstellen URLs	63
Grundlagen der Vorsignierung	64
Vorsignierung POST und Anfragen PUT	64
Eigenständiger Unterzeichner	65
Komponententests	67
Generieren Sie automatisch Mocks mithilfe von Mockall	67
Statische Wiedergabe	72
Waiter	76
Codebeispiele	78
API Gateway	79
Aktionen	80
Szenarien	81
AWS Beiträge der Gemeinschaft	82
API-Gateway-Verwaltungs-API	82
Aktionen	80
Application Auto Scaling	84
Aktionen	80
Aurora	85
Grundlagen	87
Aktionen	80
Auto Scaling	233
Grundlagen	87
Aktionen	80
Amazon Bedrock Runtime	267
Anthropic Claude	268
Amazon Cognito Identity Provider	283
Aktionen	80
Amazon Cognito Sync	285
Aktionen	80
Firehose	286
Aktionen	80
Amazon DocumentDB	287
Serverless-Beispiele	288
DynamoDB	290
Aktionen	80

Szenarien	81
Serverless-Beispiele	288
AWS Beiträge der Gemeinschaft	82
Amazon EBS	308
Aktionen	80
Amazon EC2	311
Grundlagen	87
Aktionen	80
Amazon ECR	373
Aktionen	80
Amazon ECS	375
Aktionen	80
Amazon EKS	378
Aktionen	80
AWS Glue	380
Grundlagen	87
Aktionen	80
IAM	395
Grundlagen	87
Aktionen	80
AWS IoT	422
Aktionen	80
Kinesis	424
Aktionen	80
Serverless-Beispiele	288
AWS KMS	432
Aktionen	80
Lambda	441
Grundlagen	87
Aktionen	80
Szenarien	81
Serverless-Beispiele	288
AWS Beiträge der Gemeinschaft	82
MediaLive	492
Aktionen	80
MediaPackage	493

Aktionen	80
Amazon MSK	495
Serverless-Beispiele	288
Amazon Polly	497
Aktionen	80
Szenarien	81
QLDB	502
Aktionen	80
Amazon RDS	504
Serverless-Beispiele	288
Amazon RDS Data Service	508
Aktionen	80
Amazon Rekognition	509
Szenarien	81
Route 53	511
Aktionen	80
Amazon S3	513
Grundlagen	87
Aktionen	80
Szenarien	81
Serverless-Beispiele	288
SageMaker KI	563
Aktionen	80
Secrets Manager	566
Aktionen	80
Amazon SES SES-API v2	567
Aktionen	80
Szenarien	81
Amazon SNS	583
Aktionen	80
Szenarien	81
Serverless-Beispiele	288
Amazon SQS	589
Aktionen	80
Serverless-Beispiele	288
AWS STS	595

Aktionen	80
Systems Manager	596
Aktionen	80
Amazon Transcribe	599
Szenarien	81
Sicherheit	601
Datenschutz	601
Compliance-Validierung	603
Sicherheit der Infrastruktur	604
Erzwingen Sie eine TLS-Mindestversion	605
Vom SDK verwendete Kisten	607
Smithy-Kisten	607
Mit dem SDK verwendete Kisten	607
Andere Kisten	608
Dokumentverlauf	609
.....	dcx

Was ist der AWS SDK for Rust?

Rust ist eine Systemprogrammiersprache ohne Garbage-Collector, die sich auf drei Ziele konzentriert: Sicherheit, Geschwindigkeit und Parallelität.

Das AWS SDK for Rust (das SDK) bietet Rust die Möglichkeit, mit den Infrastrukturdiensten von Amazon Web Services APIs zu interagieren. Mit dem SDK können Sie Anwendungen auf Amazon S3, Amazon EC2, DynamoDB und mehr erstellen.

Themen

- [Erste Schritte mit dem SDK](#)
- [Wartung und Support für SDK-Hauptversionen](#)
- [Weitere Ressourcen](#)

Erste Schritte mit dem SDK

Wenn Sie das SDK zum ersten Mal verwenden, empfehlen wir Ihnen, zunächst zu lesen [Fangen Sie an mit dem AWS SDK for Rust](#).

Informationen zur Konfiguration und Einrichtung, einschließlich der Erstellung und Konfiguration von Service-Clients für Anfragen AWS-Services, finden Sie unter [Konfigurieren Sie den AWS SDK for Rust](#).

Hinweise zur Verwendung des SDK finden Sie unter [Benutze die AWS SDK for Rust](#).

Eine vollständige Liste der Rust-Codebeispiele finden Sie unter [Codebeispiele](#).

Wartung und Support für SDK-Hauptversionen

Informationen zur Wartung und zum Support für SDK-Hauptversionen und die ihnen zugrunde liegenden Abhängigkeiten finden Sie im Referenzhandbuch [AWS SDKs und im Tools-Referenzhandbuch](#):

- [AWS SDKs und Richtlinien zur Wartung von Tools](#)
- [AWS SDKs Matrix zur Support von Versionen und Tools](#)

Weitere Ressourcen

Zusätzlich zu diesem Handbuch finden Sie im Folgenden wertvolle Online-Ressourcen für SDK-Entwickler:

- [AWS SDKs Referenzhandbuch für Tools und Tools](#): Enthält Einstellungen, Funktionen und andere grundlegende Konzepte, die allen gemeinsam sind. AWS SDKs
- [Website der Programmiersprache Rust](#)
- [AWS SDK for Rust API Reference](#)
- [AWS Blog zu Entwicklertools](#)
- [AWS SDK for Rust Quellcode](#) auf GitHub
- [Der AWS Codebeispielkatalog](#)

Fangen Sie an mit dem AWS SDK for Rust

Erfahren Sie, wie Sie das SDK installieren, einrichten und verwenden, um eine Rust-Anwendung für den programmgesteuerten Zugriff auf eine AWS Ressource zu erstellen.

Themen

- [SDK-Authentifizierung mit AWS](#)
- [Hallo Tutorial für die AWS SDK for Rust](#)
- [Grundlagen](#)

SDK-Authentifizierung mit AWS

Sie müssen festlegen, wie sich Ihr Code AWS bei der Entwicklung mit authentifiziert. AWS-Services Sie können den programmatischen Zugriff auf AWS Ressourcen je nach Umgebung und verfügbarem AWS Zugriff auf unterschiedliche Weise konfigurieren.

Informationen zur Auswahl Ihrer Authentifizierungsmethode und deren Konfiguration für das SDK finden Sie unter [Authentifizierung und Zugriff](#) im AWS SDKs Referenzhandbuch zu Tools.

Wir empfehlen, dass neue Benutzer, die sich lokal weiterentwickeln und von ihrem Arbeitgeber keine Authentifizierungsmethode erhalten, diese einrichten AWS IAM Identity Center. Diese Methode beinhaltet die Installation von, AWS CLI um die Konfiguration zu vereinfachen und sich regelmäßig beim AWS Zugangportal anzumelden. Wenn Sie sich für diese Methode entscheiden, sollte Ihre Umgebung die folgenden Elemente enthalten, nachdem Sie das Verfahren für die [IAM Identity Center-Authentifizierung](#) im Referenzhandbuch AWS SDKs und im Tools-Referenzhandbuch abgeschlossen haben:

- Die AWS CLI, mit der Sie eine AWS Access-Portal-Sitzung starten, bevor Sie Ihre Anwendung ausführen.
- Eine [gemeinsam genutzte AWSconfig Datei](#) mit einem [default] Profil mit einer Reihe von Konfigurationswerten, auf die vom SDK aus verwiesen werden kann. Informationen zum Speicherort dieser Datei finden Sie unter [Speicherort der gemeinsam genutzten Dateien](#) im Referenzhandbuch AWS SDKs und im Tools-Referenzhandbuch.
- Die gemeinsam genutzte config Datei legt die [region](#)Einstellung fest. Dies legt die Standardeinstellung AWS-Region fest, die das SDK für AWS Anfragen verwendet. Diese Region

wird für SDK-Dienstanforderungen verwendet, für die keine zu verwendende Region angegeben ist.

- Das SDK verwendet die [Konfiguration des SSO-Token-Anbieters](#) des Profils, um Anmeldeinformationen abzurufen, bevor Anfragen an gesendet AWS werden. Der `sso_role_name` Wert, bei dem es sich um eine IAM-Rolle handelt, die mit einem IAM Identity Center-Berechtigungssatz verbunden ist, ermöglicht den Zugriff auf die in Ihrer AWS-Services Anwendung verwendeten.

Die folgende `config` Beispieldatei zeigt ein Standardprofil, das mit der Konfiguration des SSO-Token-Anbieters eingerichtet wurde. Die `sso_session` Einstellung des Profils bezieht sich auf den genannten [sso-sessionAbschnitt](#). Der `sso-session` Abschnitt enthält Einstellungen zum Initiieren einer AWS Access-Portal-Sitzung.

```
[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
```

Das SDK für Rust benötigt keine zusätzlichen Pakete (wie `SSO` und `SSO0IDC`), die zu Ihrer Anwendung hinzugefügt werden müssen, um die IAM Identity Center-Authentifizierung verwenden zu können.

Starten Sie eine AWS Access-Portal-Sitzung

Bevor Sie eine Zugriffsanwendung ausführen AWS-Services, benötigen Sie eine aktive AWS Access-Portal-Sitzung, damit das SDK die IAM Identity Center-Authentifizierung zur Auflösung von Anmeldeinformationen verwenden kann. Abhängig von Ihrer konfigurierten Sitzungsdauer läuft Ihr Zugriff irgendwann ab und das SDK wird auf einen Authentifizierungsfehler stoßen. Um sich beim AWS Zugriffsportal anzumelden, führen Sie den folgenden Befehl in der aus AWS CLI.

```
$ aws sso login
```

Wenn Sie die Anweisungen befolgt haben und ein Standardprofil eingerichtet haben, müssen Sie den Befehl nicht mit einer `--profile` Option aufrufen. Wenn die Konfiguration Ihres SSO-Token-Anbieters ein benanntes Profil verwendet, lautet der Befehl `aws sso login --profile named-profile`.

Führen Sie den folgenden AWS CLI Befehl aus, um optional zu testen, ob Sie bereits eine aktive Sitzung haben.

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

Wenn Ihre Sitzung aktiv ist, werden in der Antwort auf diesen Befehl das in der gemeinsam genutzten `config` Datei konfigurierte IAM Identity Center-Konto und der Berechtigungssatz gemeldet.

Note

Wenn Sie bereits eine aktive AWS Access-Portal-Sitzung haben und ausführen `aws sso login`, müssen Sie keine Anmeldeinformationen angeben.

Beim Anmeldevorgang werden Sie möglicherweise aufgefordert, den AWS CLI Zugriff auf Ihre Daten zu gewähren. Da AWS CLI das auf dem SDK für Python aufbaut, können Berechtigungsnachrichten Variationen des `botocore` Namens enthalten.

Weitere Authentifizierungsinformationen

Menschliche Benutzer, auch bekannt als menschliche Identitäten, sind die Personen, Administratoren, Entwickler, Betreiber und Verbraucher Ihrer Anwendungen. Sie müssen über eine Identität verfügen, um auf Ihre AWS Umgebungen und Anwendungen zugreifen zu können. Menschliche Benutzer, die Mitglieder Ihres Unternehmens sind, also Sie, der Entwickler, werden als Personalidentitäten bezeichnet.

Verwenden Sie beim Zugriff AWS temporäre Anmeldeinformationen. Sie können einen Identitätsanbieter für Ihre menschlichen Benutzer verwenden, um Verbundzugriff auf AWS Konten zu ermöglichen, indem Sie Rollen übernehmen, die temporäre Anmeldeinformationen bereitstellen. Für eine zentralisierte Zugriffsverwaltung empfehlen wir Ihnen, AWS IAM Identity Center (IAM Identity Center) zu verwenden, um den Zugriff auf Ihre Konten und die Berechtigungen innerhalb dieser Konten zu verwalten. Weitere Alternativen finden Sie im Folgenden:

- Weitere Informationen zu bewährten Methoden finden Sie unter [Bewährte Methoden für die Sicherheit in IAM](#) im IAM-Benutzerhandbuch.

- Informationen zum Erstellen kurzfristiger AWS Anmeldeinformationen finden Sie unter [Temporäre Sicherheitsanmeldeinformationen](#) im IAM-Benutzerhandbuch.
- Weitere Informationen zu anderen Anbietern von Anmeldeinformationen, die vom SDK für Rust unterstützt werden, finden Sie unter [Standardisierte Anbieter von Anmeldeinformationen](#) im Referenzhandbuch AWS SDKs zu Tools.

Hallo Tutorial für die AWS SDK for Rust

Voraussetzungen

Um das verwenden zu können AWS SDK for Rust, müssen Sie Rust und Cargo installiert haben.

- Installieren Sie die Rust-Toolchain: <https://www.rust-lang.org/tools/install>
- Installieren Sie das cargo-component [Tool](#), indem Sie den folgenden Befehl ausführen: `cargo install cargo-component`

Empfohlene Tools:

Die folgenden optionalen Tools können in Ihrer IDE installiert werden, um Sie bei der Codevervollständigung und Fehlerbehebung zu unterstützen.

- Die Rust-Analyzer-Erweiterung finden Sie unter [rust in Visual Studio Code](#).
- Amazon Q Developer, siehe [Installation der Amazon Q Developer-Erweiterung oder des Amazon Q Developer-Plugins in Ihrer IDE](#).

Erstellen Sie Ihre erste SDK-App

Mit diesem Verfahren wird Ihre erste SDK für Rust-Anwendung erstellt, die Ihre DynamoDB-Tabellen auflistet.

1. Navigieren Sie in einem Terminal- oder Konsolenfenster zu einem Speicherort auf Ihrem Computer, an dem Sie die App erstellen möchten.
2. Führen Sie den folgenden Befehl aus, um ein `hello_world` Verzeichnis zu erstellen und es mit einem Rust-Skelettprojekt zu füllen:

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

3. Navigiere in das `hello_world` Verzeichnis und verwende den folgenden Befehl, um der App die erforderlichen Abhängigkeiten hinzuzufügen:

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

Zu diesen Abhängigkeiten gehören die SDK-Kisten, die Konfigurationsfunktionen und Unterstützung für DynamoDB bereitstellen, einschließlich der [tokioCrate](#), die zur Implementierung asynchroner I/O-Operationen verwendet wird.

Note

Sofern Sie keine Funktion wie `tokio/full` Tokio verwenden, wird keine asynchrone Laufzeit bereitgestellt. Das SDK für Rust benötigt eine asynchrone Laufzeit.

4. Aktualisieren Sie `main.rs` das `src` Verzeichnis so, dass es den folgenden Code enthält.

```
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

In diesem Beispiel wird nur die erste Ergebnisseite angezeigt. Weitere Informationen [the section called "Paginierung"](#) zum Umgang mit mehreren Ergebnisseiten finden Sie unter.

5. Führen Sie das Programm aus:

```
$ cargo run
```

Sie sollten eine Liste Ihrer Tabellennamen sehen.

Grundlagen

Voraussetzungen

Um das verwenden zu können AWS SDK for Rust, müssen Sie Rust und Cargo installiert haben.

- Installieren Sie die Rust-Toolchain: <https://www.rust-lang.org/tools/install>
- Installieren Sie das cargo-component [Tool](#), indem Sie den folgenden Befehl ausführen: `cargo install cargo-component`

Empfohlene Tools:

Die folgenden optionalen Tools können in Ihrer IDE installiert werden, um Sie bei der Codevollständigkeit und Fehlerbehebung zu unterstützen.

- Die Rust-Analyser-Erweiterung finden Sie unter [rust in Visual Studio Code](#).
- Amazon Q Developer, siehe [Installation der Amazon Q Developer-Erweiterung oder des Amazon Q Developer-Plugins in Ihrer IDE](#).

Grundlagen von Rust

Im Folgenden sind einige Grundlagen der Programmiersprache Rust aufgeführt, deren Kenntnis hilfreich wäre. Alle Referenzen für weitere Informationen stammen aus [der Programmiersprache Rust](#).

- `Cargo.toml` ist die Standard-Rust-Projektkonfigurationsdatei. Sie enthält die Abhängigkeiten und einige Metadaten über das Projekt. Rust-Quelldateien haben eine `.rs` Dateierweiterung. Siehe [Hallo, Cargo!](#).
- `Cargo.toml` Sie können mit Profilen angepasst werden, siehe [Anpassen von Builds mit Release-Profilen](#). Diese Profile haben nichts miteinander zu tun und sind unabhängig von der Verwendung AWS von Profilen in der gemeinsam genutzten AWS `config` Datei.
- Eine übliche Methode, Bibliotheksabhängigkeiten zu Ihrem Projekt und dieser Datei hinzuzufügen, ist die Verwendung `cargo add`. Siehe [cargo-add](#).
- Rust hat eine grundlegende Funktionsstruktur wie die folgende. Das `let` Schlüsselwort deklariert eine Variable und kann mit einer Zuweisung (`=`) kombiniert werden. Wenn Sie danach keinen Typ angeben `let`, leitet der Compiler davon ab. Siehe [Variablen und Veränderbarkeit](#).

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

- Um eine Variable `x` mit einem expliziten Typ zu deklarieren `T`, verwendet Rust Syntax `x : T`. Siehe [Datentypen](#).
- `struct X {}` definiert den neuen Typ `X`. Methoden werden für den benutzerdefinierten Strukturtyp implementiert `X`. Methoden für den Typ `X` werden mit Implementierungsblöcken deklariert, denen ein Schlüsselwort `impl` vorangestellt ist. `self` Bezieht sich innerhalb des Implementierungsblocks auf die Instanz der Struktur, für die die Methode aufgerufen wurde. Siehe [Syntax von Schlüsselwörtern `impl` und Methoden](#).
- Wenn ein Ausrufezeichen (`„!“`) folgt dem, was wie eine Funktionsdefinition oder ein Funktionsaufruf aussieht, dann definiert der Code ein Makro oder ruft es auf. Siehe [Makros](#).
- In Rust werden nicht behebbare Fehler durch das Makro dargestellt. `panic!` Wenn ein Programm auf ein Problem stößt, stoppt `panic!` es die Ausführung, gibt eine Fehlermeldung aus, wickelt den Vorgang ab, bereinigt den Stapel und beendet das Programm. Weitere Informationen finden Sie unter [Nicht behebbare Fehler mit `panic!`](#)

- Rust unterstützt die Vererbung von Funktionen aus Basisklassen nicht wie andere Programmiersprachen. So ermöglicht Rust das Überladen von Methoden. `traits` Man könnte sich vorstellen, dass Merkmale konzeptionell einer Schnittstelle ähneln. Merkmale und echte Schnittstellen weisen jedoch Unterschiede auf und werden im Entwurfsprozess oft unterschiedlich verwendet. Siehe [Merkmale: Definition von gemeinsamem Verhalten](#).
- Polymorphismus bezieht sich auf Code, der Funktionen für mehrere Datentypen unterstützt, ohne dass jeder einzelne Datentyp einzeln geschrieben werden muss. Rust unterstützt Polymorphismus durch Aufzählungen, Merkmale und Generika. Siehe [Vererbung als Typsystem und als Code-Sharing](#).
- Rust ist sehr explizit, wenn es um Speicher geht. Intelligente Zeiger „sind Datenstrukturen, die sich wie ein Zeiger verhalten, aber auch zusätzliche Metadaten und Funktionen haben“. Siehe [Intelligente Zeiger](#).
 - Bei diesem Typ `Cow` handelt es sich um einen clone-on-write intelligenten Zeiger, mit dessen Hilfe der Speicherbesitz bei Bedarf auf den Anrufer übertragen wird. Siehe [Enum `std::borrow::Cow`](#).
 - Bei diesem Typ `Arc` handelt es sich um einen intelligenten Zeiger mit atomarer Referenzzählung, der zugewiesene Instanzen zählt. Siehe [Struct `std::sync::Arc`](#).
- Das SDK für Rust verwendet häufig das Builder-Muster für die Konstruktion komplexer Typen.

AWS SDK for Rust Grundlagen erstellen

- Die SDK for Rust-Bibliothek ist jeweils in verschiedene Bibliothekskisten unterteilt. AWS-Service [Diese Kisten sind unter https://docs.rs/ verfügbar](https://docs.rs/).
- AWS-Service Kisten folgen der Namenskonvention von `aws-sdk-[servicename]`, z. B. `aws-sdk-s3` und `aws-sdk-dynamodb`.
- Die primäre Kernbox für die Funktionalität des SDK für Rust ist `aws-config`. Dies ist in den meisten Projekten enthalten, da es Funktionen zum Lesen von Konfigurationen aus der Umgebung bietet.
 - Verwechseln Sie das nicht mit dem AWS-Service, was genannt AWS Config wird. Da es sich um einen Dienst handelt, folgt er der Standardkonvention und wird aufgerufen `aws-sdk-config`.

Projektkonfiguration für die Arbeit mit AWS-Services

- Sie müssen Ihrem Projekt für jedes Projekt, das Ihre Anwendung verwenden soll AWS-Service , eine Kiste hinzufügen.
- Die empfohlene Methode, eine Kiste hinzuzufügen, besteht darin, die Befehlszeile im Verzeichnis Ihres Projekts zu verwenden `cargo add [crateName]`, indem Sie beispielsweise Folgendes ausführen. `cargo add aws-sdk-s3`
 - Dadurch wird dem Ordner Ihres Projekts eine Zeile hinzugefügt `Cargo.toml`. `[dependencies]`
 - Standardmäßig wird dadurch die neueste Version der Kiste zu Ihrem Projekt hinzugefügt.
- Verwenden Sie in Ihrer Quelldatei die `use` Anweisung, um Elemente aus ihren Kisten in den Gültigkeitsbereich einzubeziehen. Siehe [Verwenden externer Pakete](#) auf der Website der Programmiersprache Rust.
 - Kistennamen werden oft mit Bindestrich getrennt, aber die Bindestriche werden in Unterstriche umgewandelt, wenn die Kiste tatsächlich verwendet wird. Zum Beispiel wird die `aws-config` Kiste in der Codeanweisung als: verwendet. `use use aws_config`
- Konfiguration ist ein komplexes Thema. Die Konfiguration kann direkt im Code erfolgen oder extern in Umgebungsvariablen oder Konfigurationsdateien angegeben werden. Weitere Informationen finden Sie unter [Konfigurationsoptionen](#).
 - Wenn das SDK Ihre Konfiguration lädt, werden ungültige Werte protokolliert, anstatt die Ausführung anzuhalten, da die meisten Einstellungen vernünftige Standardwerte haben. Informationen zum Aktivieren der Protokollierung finden Sie unter. [Protokollierung von AWS SDK for Rust Code aktivieren](#)
 - Die meisten Umgebungsvariablen und Einstellungen der Konfigurationsdatei werden einmal geladen, wenn Ihr Programm gestartet wird. Aktualisierungen der Werte werden erst sichtbar, wenn Sie Ihr Programm neu starten.

Tokio-Laufzeit

- Tokio ist eine asynchrone Laufzeit für die Programmiersprache SDK für Rust, sie führt die Aufgaben aus. `async` [Siehe tokio.rs und docs.rs/tokio](#).
- Das SDK für Rust benötigt eine asynchrone Laufzeit. Wir empfehlen Ihnen, Ihren Projekten die folgende Kiste hinzuzufügen:

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

- Das `tokio::main` Attribut-Makro erstellt einen asynchronen Haupteinstiegspunkt zu Ihrem Programm. Um dieses Makro zu verwenden, fügen Sie es der Zeile vor Ihrer `main` Methode hinzu, wie im Folgenden gezeigt:

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

Konfigurieren Sie den AWS SDK for Rust

Erfahren Sie, wie Sie das AWS SDK for Rust konfigurieren. Sie müssen festlegen, wie sich Ihr Code authentifiziert AWS, wenn Sie mit AWS-Services entwickeln. Sie müssen auch festlegen, was AWS-Region Sie verwenden möchten.

Das [AWS SDKs Referenzhandbuch für Tools](#) enthält außerdem Einstellungen, Funktionen und andere grundlegende Konzepte, die in vielen der AWS SDKs Tools üblich sind.

Themen

- [Erstellen Sie einen Service-Client](#)
- [Verhaltensversionen](#)
- [Anbieter von Anmeldeinformationen](#)
- [AWS-Region Auswahl](#)
- [Konfigurationsoptionen](#)
- [Client-Endpunkte](#)
- [HTTP](#)
- [Abfangjäger](#)
- [Verwenden Sie Beobachtbarkeitsfunktionen](#)
- [Eine einzelne Betriebskonfiguration des Clients überschreiben](#)
- [Wiederholversuche](#)
- [Timeouts](#)

Erstellen Sie einen Service-Client

Um eine Anfrage an zu stellen AWS-Service, instanziiieren Sie zunächst einen Client für diesen Dienst. Sie können allgemeine Einstellungen für Service-Clients wie Timeouts und den HTTP-Client konfigurieren und die Konfiguration erneut versuchen.

Jeder Dienstclient benötigt einen Anmeldeinformationsanbieter AWS-Region und einen Anmeldeinformationsanbieter. Das SDK verwendet diese Werte, um Anfragen an die richtige Region für Ihre Ressourcen zu senden und Anfragen mit den richtigen Anmeldeinformationen zu signieren. Sie können diese Werte programmgesteuert im Code angeben oder sie automatisch aus der Umgebung laden lassen.

Note

Die Erstellung von Service-Clients kann teuer sein und sie sind in der Regel für die gemeinsame Nutzung vorgesehen. Um dies zu erleichtern, werden alle Client Strukturen implementiertClone.

Das SDK hat eine Reihe von Orten (oder Quellen), die es überprüft, um einen Wert für Konfigurationseinstellungen zu finden.

1. Jede explizite Einstellung, die im Code oder auf einem Service-Client selbst festgelegt ist, hat Vorrang vor allen anderen Einstellungen.
2. Umgebungsvariablen
 - Einzelheiten zum Setzen von Umgebungsvariablen finden Sie unter [Umgebungsvariablen](#) im Referenzhandbuch AWS SDKs und im Tools-Referenzhandbuch.
3. Geteilte credentials Dateien config und Dateien
 - Einzelheiten zum Einrichten dieser Dateien finden Sie im Referenzhandbuch „[Gemeinsam genutzte configcredentials Dateien](#)“ AWS SDKs und „Tools“.
4. Jeder vom SDK-Quellcode selbst bereitgestellte Standardwert wird zuletzt verwendet.
 - Für einige Eigenschaften, z. B. Region, gibt es keine Standardeinstellung. Sie müssen sie entweder explizit im Code, in einer Umgebungseinstellung oder in der gemeinsam genutzten config Datei angeben. Wenn das SDK die erforderliche Konfiguration nicht auflösen kann, können API-Anfragen zur Laufzeit fehlschlagen.

Die meisten Umgebungsvariableneinstellungen config sowie die Einstellungen der credentials Datei werden von mehreren AWS SDKs AND-Tools gemeinsam genutzt, um ein einheitliches Verhalten zu gewährleisten. Alle Einstellungen, die das SDK aus den Umgebungsvariablen oder Konfigurationsdateien auflösen kann, finden Sie in der Referenz zu [Einstellungen im Referenzhandbuch AWS SDKs](#) und im Tools-Referenzhandbuch.

Konfigurieren Sie einen Client aus der Umgebung

Verwenden Sie statische Methoden aus der Kiste, um einen Client mit einer Konfiguration aus der aws-config Umgebung zu erstellen:

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

```
.load()
.await;

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

Das Erstellen eines Clients auf diese Weise ist nützlich, wenn er auf Amazon Elastic Compute Cloud oder in einem anderen Kontext ausgeführt wird AWS Lambda, in dem die Konfiguration eines Service-Clients direkt in der Umgebung verfügbar ist. Dies entkoppelt Ihren Code von der Umgebung, in der er ausgeführt wird, und macht es einfacher, Ihre Anwendung für mehrere bereitzustellen, AWS-Regionen ohne den Code zu ändern.

Sie können bestimmte Eigenschaften explizit überschreiben. Die explizite Konfiguration hat Vorrang vor der Konfiguration, die in der Ausführungsumgebung aufgelöst wurde. Das folgende Beispiel lädt die Konfiguration aus der Umgebung, überschreibt jedoch ausdrücklich: AWS-Region

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

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

Note

Nicht alle Konfigurationswerte stammen zum Zeitpunkt der Erstellung vom Client. Auf Einstellungen im Zusammenhang mit Anmeldeinformationen, wie z. B. temporäre Zugriffsschlüssel und die IAM Identity Center-Konfiguration, wird über die Ebene des Anmeldeinformationsanbieters zugegriffen, wenn der Client für eine Anfrage verwendet wird.

Der in den vorherigen Beispielen `BehaviorVersion::latest()` gezeigte Code gibt die Version des SDK an, die standardmäßig verwendet werden soll. `BehaviorVersion::latest()` ist für die meisten Fälle geeignet. Details hierzu finden Sie unter [Verhaltensversionen](#).

Verwenden Sie das Builder-Muster für dienstspezifische Einstellungen

Es gibt einige Optionen, die nur für einen bestimmten Service-Client-Typ konfiguriert werden können. In den meisten Fällen möchten Sie jedoch immer noch den Großteil der Konfiguration aus der Umgebung laden und dann die zusätzlichen Optionen speziell hinzufügen. Das Builder-Muster

ist ein gängiges Muster innerhalb der AWS SDK for Rust Crates. Sie laden zuerst die allgemeine Konfiguration mit und verwenden dann die `from` Methode `aws_config::defaults`, um diese Konfiguration in den Builder für den Dienst zu laden, mit dem Sie arbeiten. Anschließend können Sie beliebige eindeutige Konfigurationswerte für diesen Dienst festlegen und aufrufen `build`. Schließlich wird der Client anhand dieser modifizierten Konfiguration erstellt.

```
// 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);
```

Eine Möglichkeit, zusätzliche Methoden zu finden, die für einen bestimmten Typ von Service-Client verfügbar sind, besteht darin, die API-Dokumentation zu verwenden, z. B. für [aws_sdk_s3::config::Builder](#).

Erweiterte explizite Client-Konfiguration

Um einen Service-Client mit bestimmten Werten zu konfigurieren, anstatt eine Konfiguration aus der Umgebung zu laden, können Sie diese im Client Config Builder angeben, wie im Folgenden gezeigt:

```
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);
```

Wenn Sie eine Dienstkonfiguration mit erstellen `aws_sdk_s3::Config::builder()`, wird keine Standardkonfiguration geladen. Standardwerte werden nur geladen, wenn eine Konfiguration erstellt wird, die auf `aws_config::defaults` basiert.

Es gibt einige Optionen, die nur für einen bestimmten Service-Client-Typ konfiguriert werden können. Das vorherige Beispiel zeigt ein Beispiel dafür, indem die `endpoint_resolver` Funktion auf einem Amazon S3 S3-Client verwendet wird.

Verhaltensversionen

AWS SDK for Rust Entwickler erwarten das robuste und vorhersehbare Verhalten, das die Sprache und ihre wichtigsten Bibliotheken bieten, und verlassen sich darauf. Um Entwicklern, die das SDK für Rust verwenden, dabei zu helfen, das erwartete Verhalten zu erzielen, müssen Client-Konfigurationen `a` enthalten `BehaviorVersion`. Das `BehaviorVersion` gibt die Version des SDK an, deren Standardwerte erwartet werden. Auf diese Weise kann sich das SDK im Laufe der Zeit weiterentwickeln und bewährte Verfahren ändern, um neuen Standards zu entsprechen und neue Funktionen zu unterstützen, ohne dass sich dies unerwartet negativ auf das Verhalten Ihrer Anwendung auswirkt.

Warning

Wenn Sie versuchen, das SDK zu konfigurieren oder einen Client zu erstellen, ohne explizit `a` anzugeben `BehaviorVersion`, wird der Konstruktor `panic`.

Stellen Sie sich zum Beispiel vor, dass eine neue Version des SDK mit einer neuen Standard-Wiederholungsrichtlinie veröffentlicht wird. Wenn Ihre Anwendung eine `BehaviorVersion` mit einer früheren Version des SDK übereinstimmende Version verwendet, wird diese vorherige Konfiguration anstelle der neuen Standardkonfiguration verwendet.

Jedes Mal, wenn eine neue Verhaltensversion des SDK für Rust veröffentlicht wird, `BehaviorVersion` wird die vorherige Version mit dem SDK für `deprecated` Rust-Attribut gekennzeichnet und die neue Version wird hinzugefügt. Dies führt dazu, dass bei der Kompilierung Warnungen auftreten, ansonsten kann der Build aber wie gewohnt fortgesetzt werden. `BehaviorVersion::latest()` wird ebenfalls aktualisiert, um das Standardverhalten der neuen Version anzuzeigen.

Note

In den meisten Fällen sollten Sie `BehaviorVersion::latest()` im Code oder das Feature-Flag `behavior-version-latest` in der `Cargo.toml` Datei verwenden. Es wird empfohlen, sich nur so lange an eine bestimmte Version zu binden, wie es erforderlich ist.

Stellen Sie die Verhaltensversion ein `Cargo.toml`

Sie können die Verhaltensversion für das SDK und einzelne Module, z. B. `aws-sdk-s3` oder, `aws-sdk-iam`, indem Sie der `Cargo.toml` Datei ein entsprechendes Feature-Flag hinzufügen. Derzeit wird nur die `latest` Version des SDK unterstützt in `Cargo.toml`:

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

Legen Sie die Verhaltensversion im Code fest

Ihr Code kann die Verhaltensversion nach Bedarf ändern, indem Sie sie bei der Konfiguration des SDK oder eines Clients angeben:

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

In diesem Beispiel wird eine Konfiguration erstellt, die die Umgebung verwendet, um das SDK zu konfigurieren, aber den Wert `BehaviorVersion` auf `setzv2023_11_09()`.

Anbieter von Anmeldeinformationen

Um Anfragen zur AWS Verwendung von zu stellen AWS SDK for Rust, verwendet das SDK kryptografisch signierte Anmeldeinformationen, ausgestellt von. AWS Zur Laufzeit ruft das SDK Konfigurationswerte für Anmeldeinformationen ab, indem es mehrere Speicherorte überprüft.

Wenn die abgerufene Konfiguration [Einstellungen für den AWS IAM Identity Center Single Sign-On-Zugriff](#) enthält, ruft das SDK zusammen mit dem IAM Identity Center temporäre Anmeldeinformationen ab, an die es Anfragen sendet. AWS-Services

Wenn die abgerufene Konfiguration [temporäre Anmeldeinformationen](#) enthält, verwendet das SDK diese, um Aufrufe zu tätigen AWS-Service . Temporäre Anmeldeinformationen bestehen aus Zugriffsschlüsseln und einem Sitzungstoken.

Die Authentifizierung mit AWS kann außerhalb Ihrer Codebasis erfolgen. Viele Authentifizierungsmethoden können vom SDK mithilfe der Credential Provider-Kette automatisch erkannt, verwendet und aktualisiert werden.

Anleitungen zu den ersten Schritten mit der AWS Authentifizierung für Ihr Projekt finden Sie unter [Authentifizierung und Zugriff](#) im AWS SDKs Referenzhandbuch zu Tools.

Die Kette der Anbieter von Anmeldeinformationen

Wenn Sie bei der Erstellung eines Clients nicht explizit einen Anbieter für Anmeldeinformationen angeben, verwendet das SDK für Rust eine Kette von Anbietern von Anmeldeinformationen, die eine Reihe von Stellen überprüft, an denen Sie Anmeldeinformationen angeben können. Sobald das SDK Anmeldeinformationen an einem dieser Orte findet, wird die Suche beendet. Einzelheiten zur Erstellung von Clients finden Sie unter [Erstellen Sie einen Service-Client](#).

Im folgenden Beispiel wird kein Anbieter für Anmeldeinformationen im Code angegeben. Das SDK verwendet die Anmeldeinformationsanbieterkette, um die Authentifizierung zu erkennen, die in der Hostumgebung eingerichtet wurde, und verwendet diese Authentifizierung für Aufrufe von AWS-Services

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

Reihenfolge beim Abrufen der Anmeldeinformationen

Die Anmeldeinformationsanbieterkette sucht anhand der folgenden vordefinierten Reihenfolge nach Anmeldeinformationen:

1. Greifen Sie auf wichtige Umgebungsvariablen zu

Das SDK versucht, Anmeldeinformationen aus den `AWS_ACCESS_KEY_ID` `AWS_SESSION_TOKEN` Umgebungsvariablen und zu laden. `AWS_SECRET_ACCESS_KEY`

2. Die geteilten **credentials** Dateien `AWS config` und

Das SDK versucht, Anmeldeinformationen aus dem [default] Profil in die gemeinsam genutzten `credentials` Dateien `AWS config` und Dateien zu laden. Sie können die

AWS_PROFILE Umgebungsvariable verwenden, um ein benanntes Profil auszuwählen, das das SDK laden und nicht verwenden soll[default]. Die credentials Dateien config und werden von verschiedenen AWS SDKs AND-Tools gemeinsam genutzt. Weitere Informationen zu diesen Dateien finden Sie unter [Gemeinsam genutzte credentials Dateien config und Dateien](#) im AWS SDKs Referenzhandbuch für Tools.

Wenn Sie IAM Identity Center zur Authentifizierung verwenden, verwendet das SDK für Rust in diesem Fall das Single Sign-On-Token, das durch Ausführen AWS des CLI-Befehls eingerichtet wurde. `aws sso login` Das SDK verwendet die temporären Anmeldeinformationen, die das IAM Identity Center gegen ein gültiges Token ausgetauscht hat. Das SDK verwendet dann die temporären Anmeldeinformationen, wenn es aufruft AWS-Services. Ausführliche Informationen zu diesem Prozess finden Sie unter [Grundlegendes zur Auflösung von SDK-Anmeldeinformationen](#) im Referenzhandbuch AWS SDKs und AWS-Services im Tools-Referenzhandbuch.

- Anleitungen zur Konfiguration dieses Anbieters finden Sie unter [IAM Identity Center-Authentifizierung](#) im Referenzhandbuch AWS SDKs und im Tools-Referenzhandbuch.
- Einzelheiten zu den SDK-Konfigurationseigenschaften für diesen Anbieter finden Sie unter [IAM Identity Center-Anmeldeinformationsanbieter](#) im Referenzhandbuch AWS SDKs und im Tools-Referenzhandbuch.

3. AWS STS Webidentität

Beim Erstellen von mobilen Anwendungen oder clientbasierten Webanwendungen, für die Zugriff erforderlich ist AWS, gibt AWS Security Token Service (AWS STS) einen Satz temporärer Sicherheitsanmeldedaten für Verbundbenutzer zurück, die über einen Public Identity Provider (IdP) authentifiziert wurden.

- Wenn Sie dies in einem Profil angeben, versucht das SDK oder das Tool, temporäre Anmeldeinformationen mithilfe der API-Methode abzurufen. AWS STS `AssumeRoleWithWebIdentity` Einzelheiten zu dieser Methode finden Sie [AssumeRoleWithWebIdentity](#) in der AWS Security Token Service API-Referenz.
- Anleitungen zur Konfiguration dieses Anbieters finden Sie unter [Federate with Web Identity oder OpenID Connect](#) im AWS SDKs und Tools Reference Guide.
- Einzelheiten zu den SDK-Konfigurationseigenschaften für diesen Anbieter finden Sie unter `Assume` [role Credential Provider](#) im Referenzhandbuch AWS SDKs und im Tools-Referenzhandbuch.

4. Anmeldeinformationen für Amazon ECS- und Amazon EKS-Container

Ihren Amazon Elastic Container Service-Aufgaben und Kubernetes-Servicekonten kann eine IAM-Rolle zugewiesen werden. Die in der IAM-Rolle gewährten Berechtigungen werden von den Containern übernommen, die in der Aufgabe oder den Containern des Pods ausgeführt werden. Diese Rolle ermöglicht es Ihrem SDK für Rust-Anwendungscode (auf dem Container), andere AWS-Services zu verwenden.

Das SDK versucht, Anmeldeinformationen aus den `AWS_CONTAINER_CREDENTIALS_FULL_URI` Umgebungsvariablen `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` oder abzurufen, die automatisch von Amazon ECS und Amazon EKS festgelegt werden können.

- Einzelheiten zur Einrichtung dieser Rolle für Amazon ECS finden Sie unter [Amazon ECS-Aufgaben-IAM-Rolle](#) im Amazon Elastic Container Service Developer Guide.
- Informationen zur Einrichtung von Amazon EKS finden Sie unter [Einrichten des Amazon EKS Pod Identity Agent](#) im Amazon EKS-Benutzerhandbuch.
- Einzelheiten zu den SDK-Konfigurationseigenschaften für diesen Anbieter finden Sie unter [Container Credential Provider](#) im AWS SDKs und Tools Reference Guide.

5. Metadaten-Service für EC2 Amazon-Instances

Erstellen Sie eine IAM-Rolle und fügen Sie sie Ihrer Instance hinzu. Das SDK für die Rust-Anwendung auf der Instanz versucht, die von der Rolle bereitgestellten Anmeldeinformationen aus den Instanz-Metadaten abzurufen.

- Das SDK für Rust unterstützt nur [IMDSv2](#).
- Einzelheiten zur Einrichtung dieser Rolle und zur Verwendung von Metadaten, [IAM-Rollen für Amazon EC2](#) und [Work with Instance-Metadaten](#) finden Sie im EC2 Amazon-Benutzerhandbuch.
- Einzelheiten zu den SDK-Konfigurationseigenschaften für diesen Anbieter finden Sie unter [IMDS-Anmeldeinformationsanbieter](#) im Referenzhandbuch AWS SDKs und im Tools-Referenzhandbuch.

6. Wenn die Anmeldeinformationen zu diesem Zeitpunkt immer noch nicht geklärt sind, ist der Vorgang panics mit einem Fehler.

Einzelheiten zu den Konfigurationseinstellungen des AWS Anmeldeinformationsanbieters finden Sie unter [Standardisierte Anmeldeinformationsanbieter](#) in der Einstellungsreferenz des AWS SDKs und im Tools-Referenzhandbuch.

Anbieter für explizite Anmeldeinformationen

Anstatt sich bei der Erkennung Ihrer Authentifizierungsmethode auf die Kette der Anmeldeinformationsanbieter zu verlassen, können Sie einen bestimmten Anmeldeinformationsanbieter angeben, den das SDK verwenden soll. Wenn Sie Ihre allgemeine Konfiguration mit `ladenaws_config::defaults`, können Sie einen benutzerdefinierten Anbieter für Anmeldeinformationen angeben, wie im Folgenden dargestellt:

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

Sie können Ihren eigenen Anbieter für Anmeldeinformationen implementieren, indem Sie das [ProvideCredentials](#) Merkmal implementieren.

Zwischenspeichern von Identitäten

Das SDK speichert Anmeldeinformationen und andere Identitätstypen wie SSO-Token im Cache. Standardmäßig verwendet das SDK eine Lazy-Cache-Implementierung, die Anmeldeinformationen bei der ersten Anfrage lädt, zwischenspeichert und dann versucht, sie bei einer weiteren Anfrage zu aktualisieren, wenn sie fast ablaufen. Clients, die auf derselben Grundlage erstellt wurden, teilen SdkConfig sich eine. [IdentityCache](#)

AWS-Region Auswahl

Sie können auf diejenigen zugreifen AWS-Services , die in einem bestimmten geografischen Gebiet tätig sind, indem Sie AWS-Regionen Dies kann sowohl aus Gründen der Redundanz als auch dafür nützlich sein, dass Ihre Daten und Anwendungen in der Nähe ausgeführt werden, wo Sie und Ihre Benutzer darauf zugreifen. Weitere Informationen zur Verwendung von Regionen finden Sie [AWS-Region](#) im Referenzhandbuch AWS SDKs und im Tools-Referenzhandbuch.

Important

Die meisten Ressourcen befinden sich in einer bestimmten Region, AWS-Region und Sie müssen die richtige Region für die Ressource angeben, wenn Sie das SDK verwenden.

AWS-Region Anbieterkette

Der folgende Suchvorgang wird verwendet, wenn die Konfiguration eines Service-Clients aus der Ausführungsumgebung geladen wird. Der erste Wert, den das SDK feststellt, wird in der Konfiguration des Clients verwendet. Weitere Informationen zum Erstellen von Service-Clients finden Sie unter [Konfigurieren Sie einen Client aus der Umgebung](#).

1. Jede explizite Region, die programmgesteuert festgelegt wurde.
2. Die Umgebungsvariable `AWS_REGION` wird geprüft.
 - Wenn Sie den AWS Lambda Dienst verwenden, wird diese Umgebungsvariable automatisch vom AWS Lambda Container festgelegt.
3. Die `region` Eigenschaft in der gemeinsam genutzten `AWS config` Datei wird überprüft.
 - Die `AWS_CONFIG_FILE` Umgebungsvariable kann verwendet werden, um den Speicherort der gemeinsam genutzten `config` Datei zu ändern. Weitere Informationen darüber, wo diese Datei gespeichert wird, finden Sie unter [Speicherort der gemeinsam genutzten `credentials` Dateien `config` und Dateien](#) im AWS SDKs Referenzhandbuch zu Tools.
 - Die `AWS_PROFILE` Umgebungsvariable kann verwendet werden, um ein benanntes Profil anstelle des Standardprofils auszuwählen. Weitere Informationen zur Konfiguration verschiedener Profile finden Sie unter [Geteilte Profile `config` und `credentials` Dateien](#) im AWS SDKs Referenzhandbuch zu Tools.
4. Das SDK versucht, den Amazon EC2 Instance Metadata Service zu verwenden, um die Region der aktuell laufenden EC2 Amazon-Instance zu ermitteln.
 - Die AWS SDK for Rust einzige Unterstützung `IMDSv2`.

Konfigurationsoptionen

Viele Konfigurationseinstellungen können außerhalb Ihres Codes bearbeitet werden. Die meisten Konfigurationseinstellungen können entweder als Umgebungsvariablen oder in einer separaten gemeinsam genutzten `AWS config` Datei festgelegt werden. Die gemeinsam genutzte `config` Datei kann separate Einstellungssätze, sogenannte Profile, enthalten, um unterschiedliche Konfigurationen für verschiedene Umgebungen oder Tests bereitzustellen.

Umgebungsvariablen und gemeinsam genutzte `config` Dateieinstellungen sind standardisiert und werden von allen AWS SDKs Tools gemeinsam genutzt, um konsistente Funktionen in verschiedenen Sprachen zu gewährleisten.

Weitere Informationen zur Konfiguration Ihrer Anwendung mit diesen Methoden AWS SDKs sowie Einzelheiten zu den einzelnen SDK-übergreifenden Einstellungen finden Sie im Referenzhandbuch und im Tools-Referenzhandbuch. Einige Seiten könnten von Interesse sein:

- [Geteilte credentials Dateien config und Dateien](#) — Erläutert Profile und das Format dieser Konfigurationsdateien.
- [Speicherort der gemeinsam genutzten credentials Dateien config und Dateien](#) — Erläutert den Standardspeicherort dieser Dateien und wie dieser geändert werden kann.
- [Unterstützung von Umgebungsvariablen](#) — Erläutert die Einstellung von Umgebungsvariablen.
- [Einstellungsreferenz](#) — Referenzinformationen zu allen SDK-übergreifenden Einstellungen.

Client-Endpunkte

Wenn der AWS SDK for Rust Anruf erfolgt AWS-Service, besteht einer der ersten Schritte darin, zu bestimmen, wohin die Anfrage weitergeleitet werden soll. Dieser Vorgang wird als Endpunktauflösung bezeichnet.

Sie können die Endpunktauflösung für das SDK konfigurieren, wenn Sie einen Service-Client erstellen. Die Standardkonfiguration für die Endpunktauflösung ist normalerweise in Ordnung, aber es gibt mehrere Gründe, warum Sie die Standardkonfiguration möglicherweise ändern möchten. Zwei Beispielgründe lauten wie folgt:

- Um Anfragen an eine Vorabversion eines Dienstes oder an eine lokale Bereitstellung eines Dienstes zu stellen.
- Um auf bestimmte Servicefunktionen zuzugreifen, die noch nicht im SDK modelliert wurden.

Warning

Die Endpunktauflösung ist ein SDK-Thema für Fortgeschrittene. Wenn Sie die Standardeinstellungen ändern, riskieren Sie, dass Ihr Code beschädigt wird. Die Standardeinstellungen gelten für die meisten Benutzer in Produktionsumgebungen.

Benutzerdefinierte Endpunkte können global festgelegt werden, sodass sie für alle Serviceanfragen verwendet werden, oder Sie können einen benutzerdefinierten Endpunkt für einen bestimmten AWS-Service Endpunkt festlegen.

Benutzerdefinierte Endpunkte können mithilfe von Umgebungsvariablen oder Einstellungen in der gemeinsam genutzten `AWS config` Datei konfiguriert werden. Informationen zu diesem Ansatz finden Sie unter [Dienstspezifische Endpunkte](#) im Referenzhandbuch AWS SDKs und im Tools-Referenzhandbuch. Eine vollständige Liste der Einstellungen für gemeinsam genutzte `config` Dateien und Umgebungsvariablen für alle AWS-Services finden Sie unter [Identifikatoren für dienstspezifische Endpunkte](#).

Alternativ kann diese Anpassung auch in Ihrem Code konfiguriert werden, wie in den folgenden Abschnitten gezeigt.

Benutzerdefinierte Konfiguration

Sie können die Endpunktauflösung eines Service-Clients mit zwei Methoden anpassen, die bei der Erstellung des Clients verfügbar sind:

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

Sie können beide Eigenschaften festlegen. In den meisten Fällen geben Sie jedoch nur eine an. Für den allgemeinen Gebrauch `endpoint_url` wird es am häufigsten angepasst.

Endpunkt-URL festlegen

Sie können einen Wert für `endpoint_url` festlegen, um einen „Basis-Hostnamen“ für den Dienst anzugeben. Dieser Wert ist jedoch nicht endgültig, da er als Parameter an die `ResolveEndpoint` Instanz des Clients übergeben wird. Die `ResolveEndpoint` Implementierung kann diesen Wert dann überprüfen und möglicherweise ändern, um den endgültigen Endpunkt zu bestimmen.

Stellen Sie den Endpunkt-Resolver ein

Die `ResolveEndpoint` Implementierung eines Service-Clients bestimmt den endgültigen aufgelösten Endpunkt, den das SDK für eine bestimmte Anfrage verwendet. Ein Service-Client ruft die `resolve_endpoint` Methode für jede Anfrage auf und verwendet den vom Resolver zurückgegebenen [EndpointFuture](#) Wert ohne weitere Änderungen.

Das folgende Beispiel zeigt die Bereitstellung einer benutzerdefinierten Endpoint-Resolver-Implementierung für einen Amazon S3 S3-Client, die pro Phase einen anderen Endpunkt auflöst, z. B. Staging und Produktion:

```
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-Resolver und damit auch das `ResolveEndpoint` Merkmal sind für jeden Service spezifisch und können daher nur in der Service-Client-Konfiguration konfiguriert werden. Die Endpunkt-URL kann dagegen entweder mithilfe einer gemeinsamen Konfiguration (gilt für alle daraus abgeleiteten Dienste) oder für einen bestimmten Dienst konfiguriert werden.

ResolveEndpoint Parameter

Die `resolve_endpoint` Methode akzeptiert dienstspezifische Parameter, die Eigenschaften enthalten, die bei der Endpunktauflösung verwendet werden.

Jeder Dienst umfasst die folgenden Basiseigenschaften:

Name	Typ	Beschreibung
region	String	Die des Kunden AWS-Region
endpoint	String	Eine Zeichenkettendarstellung des Wertesatzes von endpointUrl
use_fips	Boolesch	Ob FIPS-Endpunkte in der Konfiguration des Clients aktiviert sind
use_dual_stack	Boolesch	Ob Dual-Stack-Endpunkte in der Konfiguration des Clients aktiviert sind

AWS-Services kann zusätzliche Eigenschaften angeben, die für die Auflösung erforderlich sind. Zu den Amazon S3 [S3-Endpunktparametern](#) gehören beispielsweise der Bucket-Name und mehrere Amazon S3-spezifische Funktionseinstellungen. Die `force_path_style` Eigenschaft bestimmt beispielsweise, ob die virtuelle Host-Adressierung verwendet werden kann.

Wenn Sie Ihren eigenen Anbieter implementieren, sollten Sie keine eigene Instanz von Endpunktparametern erstellen müssen. Das SDK stellt die Eigenschaften für jede Anfrage bereit und übergibt sie an Ihre Implementierung von `resolve_endpoint`.

Vergleichen Sie das Verwenden mit `endpoint_url` dem Verwenden `endpoint_resolver`

Es ist wichtig zu verstehen, dass die folgenden beiden Konfigurationen, die eine verwendet `endpoint_url` und die andere, die verwendet `endpoint_resolver`, NICHT zu Clients mit gleichwertigem Verhalten bei der Endpunktauflösung führen.

```
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();
```

Der Client, der das `endpoint_url` festlegt, gibt eine Basis-URL an, die an den (Standard-) Anbieter übergeben wird und im Rahmen der Endpunktauflösung geändert werden kann.

Der Client, der das festlegt, `endpoint_resolver` gibt die endgültige URL an, die der Amazon S3 S3-Client verwendet.

Beispiele

Benutzerdefinierte Endpunkte werden häufig zum Testen verwendet. Anstatt Anrufe an den cloudbasierten Dienst zu tätigen, werden Anrufe an einen lokal gehosteten, simulierten Dienst weitergeleitet. Zwei dieser Optionen sind:

- [DynamoDB local](#) — eine lokale Version des Amazon DynamoDB-Service.
- [LocalStack](#) — ein Cloud-Service-Emulator, der in einem Container auf Ihrem lokalen Computer ausgeführt wird.

Die folgenden Beispiele veranschaulichen zwei verschiedene Möglichkeiten, einen benutzerdefinierten Endpunkt anzugeben, um diese beiden Testoptionen zu verwenden.

DynamoDB local direkt im Code verwenden

Wie in den vorherigen Abschnitten beschrieben, können Sie `endpoint_url` direkt im Code festlegen, dass der Basisendpunkt so überschrieben wird, dass er auf den lokalen DynamoDB-Server verweist. In Ihrem Code:

```
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);
```

Das [vollständige Beispiel](#) ist verfügbar unter GitHub.

LocalStack Verwenden Sie die **config** Datei

Sie können [dienstspezifische Endpunkte](#) in Ihrer gemeinsam genutzten AWS config Datei festlegen. Das folgende Konfigurationsprofil legt fest, `endpoint_url` dass eine Verbindung zu einem Port hergestellt werden soll `localhost. 4566` Weitere Informationen zur LocalStack Konfiguration finden Sie unter [Zugriff LocalStack über die Endpunkt-URL](#) auf der LocalStack Docs-Website.

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

Das SDK nimmt die Änderungen in der gemeinsam genutzten config Datei auf und wendet sie auf Ihre SDK-Clients an, wenn Sie das `localstack` Profil verwenden. Bei diesem Ansatz muss Ihr Code keinen Verweis auf Endpunkte enthalten und würde wie folgt aussehen:

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

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

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

Das [vollständige Beispiel](#) ist verfügbar unter GitHub.

HTTP

Das AWS SDK für Rust bietet die folgenden HTTP-bezogenen Konfigurationen:

Themen

- [Hyper 1.x](#)

Hyper 1.x

Note

Das AWS SDK for Rust wurde vor der `hyper` Veröffentlichung einer stabilen 1.0-Version entwickelt. Das SDK verwendet derzeit standardmäßig die Version v0.14 von `hyper`.

`hyper` ist eine HTTP-Bibliothek auf niedrigerer Ebene für Rust, die zusammen mit der `aws-smithy-http` verwendet werden kann, um API-Dienstaufrufe AWS SDK for Rust zu tätigen. Die Referenzdokumentation zu dieser Kiste finden Sie unter [hyper](#) - Rust. Standardmäßig verwendet das SDK für Rust die Version v0.14 von `hyper` und einen Anbieter. `CryptoMode: Ring`

Clients mit `hyper 1.0`-Unterstützung befinden sich in der [aws-smithy-experimental](#) Kiste.

Wählen Sie einen Krypto-Anbieter

Sie müssen eine auswählen `CryptoProvider` und die erforderliche Funktion in der `aws-smithy-experimental` Kiste aktivieren.

Die folgenden `CryptoProvider` Modi sind verfügbar:

- `CryptoMode: AwsLc`- ein kryptografischer Anbieter, der auf [aws-lc-rs](#) basiert. Dies ist eine allgemeine kryptografische Bibliothek, die vom AWS Cryptography-Team AWS und seinen Kunden verwaltet wird und auf Code aus dem Google BoringSSL-Projekt und dem OpenSSL-Projekt basiert.
 - Erfordert `crypto-aws-lc` eine Funktion.
- `CryptoMode: AwsLcFips`- ein FIPS-konformer kryptografischer Anbieter, der auf [aws-lc-rs](#) basiert.
 - `crypto-aws-lc-fips` Erfordert eine Funktion.

Note

Dies erfordert möglicherweise zusätzliche Build-Tools, um kompilieren zu können. Weitere Informationen finden Sie im [aws-lc-rs](#) Repository und in den Build-Anweisungen.

- `CryptoMode:Ring`- ein kryptografischer Anbieter, der auf [ring](#) basiert. Dies ist eine „sichere, schnelle, kleine Krypto mit Rust“.
- Erfordert eine Funktion `crypto-ring`.

Note

Dies ist der Kryptoanbieter, den das SDK für Rust im `hyper` Standard-v0.14-Client verwendet.

Wie aktiviere ich eine Funktion für eine Kiste

Das folgende Beispiel zeigt, wie Sie die `crypto-aws-lc` Funktion zur `aws-smithy-experimental` Kiste in Ihrem Projekt hinzufügen: `Cargo.toml`

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

Alternativ können Sie den folgenden Befehl an einer Befehlszeile im Ordner Ihres Projekts ausführen:

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

Weitere Informationen finden Sie unter [Features](#) in The Cargo Book.

Erstellen Sie einen **hyper** 1.0-Client und verwenden Sie ihn mit dem SDK

Um einen `hyper` 1.0-Client zu erstellen `Cargo.toml`, fügen Sie in Ihrem Projekt die `aws-smithy-experimental` Kiste hinzu. `HyperClientBuilder` Dazu müssen Sie eine `CryptoProvider` auswählen. Im vorherigen Abschnitt wurden die Anbieteroptionen und die Aktivierung einer der kryptobezogenen Funktionen von The `aws-smithy-experimental` Crate behandelt.

Der folgende Code wird `CryptoMode::AwsLc` als Kryptoanbieter für einen `hyper 1.0`-HTTP-Client verwendet, der mit einem Amazon Simple Storage Service (Amazon S3) -Serviceclient verwendet wird. In diesem Beispiel muss Ihr Projekt `Cargo.toml` auch die AWS-Service Crate, `aws-sdk-s3` enthalten.

```
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);
```

Der HTTP-Client kann in der gemeinsamen Konfigurationsstruktur oder in der Konfigurationsstruktur eines bestimmten Dienstes festgelegt werden.

Abfangjäger

Sie können Interzeptoren verwenden, um sich an der Ausführung von API-Anfragen und -Antworten zu orientieren. Interzeptoren sind offene Mechanismen, bei denen das SDK Code aufruft, den Sie schreiben, um Verhalten in den Anforderungs-/Antwort-Lebenszyklus einzufügen. Auf diese Weise können Sie eine In-Flight-Anfrage ändern, die Anforderungsverarbeitung debuggen, Fehler anzeigen und vieles mehr.

Das folgende Beispiel zeigt einen einfachen Interceptor, der allen ausgehenden Anfragen einen zusätzlichen Header hinzufügt, bevor die Wiederholungsschleife aufgerufen wird:

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



```
use aws_smithy_runtime_api::box_error::BoxError;

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

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

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

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

        Ok(())
    }
}
```

[Weitere Informationen und die verfügbaren Interceptor-Hooks finden Sie im Trait Intercept.](#)

Registrierung des Interceptors

Sie registrieren Interzeptoren, wenn Sie einen Service-Client erstellen oder wenn Sie die Konfiguration für einen bestimmten Vorgang überschreiben. Die Registrierung ist unterschiedlich, je nachdem, ob Sie möchten, dass der Interceptor für alle Operationen Ihres Clients oder nur für bestimmte Operationen gilt.

Interceptor für alle Operationen auf einem Service-Client

Um einen Interceptor für den gesamten Client zu registrieren, fügen Sie den Interceptor mithilfe des Musters hinzu. `Builder`

```
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 nur für eine bestimmte Operation

Um einen Interceptor nur für eine einzelne Operation zu registrieren, verwenden Sie die Erweiterung `customize`. Mit dieser Methode können Sie Ihre Service-Client-Konfigurationen auf der Ebene der einzelnen Operationen überschreiben. Weitere Informationen zu anpassbaren Vorgängen finden Sie unter [Eine einzelne Betriebskonfiguration des Clients überschreiben](#).

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

Verwenden Sie Beobachtbarkeitsfunktionen

Beobachtbarkeit ist das Ausmaß, in dem der aktuelle Zustand eines Systems aus den von ihm ausgegebenen Daten abgeleitet werden kann. Die ausgegebenen Daten werden allgemein als Telemetrie bezeichnet.

Themen

- [Protokollierung von AWS SDK for Rust Code aktivieren](#)

Protokollierung von AWS SDK for Rust Code aktivieren

Das AWS SDK for Rust verwendet das [Tracing-Framework](#) für die Protokollierung.

1. Fügen Sie in einer Befehlszeile für Ihr Projektverzeichnis die [Tracing-Subscriber-Crate](#) als Abhängigkeit hinzu:

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

Dadurch wird die Kiste dem `[dependencies]` Abschnitt Ihrer Datei hinzugefügt. `Cargo.toml`

2. Initialisieren Sie den Abonnenten. Normalerweise erfolgt dies zu Beginn der `main` Funktion, bevor ein SDK für Rust-Operationen aufgerufen wird:

```
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. Schalten Sie die Protokollierung mithilfe der `RUST_LOG` Umgebungsvariablen ein. Um die Anzeige von Protokollinformationen zu aktivieren, setzen Sie in einer Befehlszeile die `RUST_LOG` Umgebungsvariable auf die Ebene, auf der Sie protokollieren möchten. Im folgenden Beispiel wird die Protokollierung auf die `debug` Stufe eingestellt:

Linux/macOS

```
$ RUST_LOG=debug
```

Windows

Wenn Sie verwenden VSCode, ist das Terminalfenster häufig standardmäßig auf PowerShell eingestellt. Überprüfen Sie, welche Art von Aufforderung Sie verwenden.

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

PowerShell

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

4. Führen Sie das Programm aus:

```
$ cargo run
```

Sie sollten zusätzliche Ausgaben in der Konsole oder im Terminalfenster sehen.

Weitere Informationen finden Sie in der `tracing-subscriber` Dokumentation unter [Filtern von Ereignissen mit Umgebungsvariablen](#).

Interpretieren Sie die Protokollausgabe

Sobald Sie die Protokollierung aktiviert haben, indem Sie die Schritte im vorherigen Abschnitt ausgeführt haben, werden zusätzliche Protokollinformationen standardmäßig ausgedruckt.

Wenn Sie das standardmäßige Protokollausgabeformat (vom Protokollierungsmodul als „vollständig“ bezeichnet) verwenden, sehen die Informationen, die Sie in der Protokollausgabe sehen, in etwa so aus:

```
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
```

Jeder Eintrag beinhaltet Folgendes:

- Der Zeitstempel des Protokolleintrags.
- Die Protokollebene des Eintrags. Das ist ein Wort wie INFODEBUG, oderTRACE.
- Die verschachtelte Gruppe von [Bereichen](#), aus denen der Protokolleintrag generiert wurde, getrennt durch Doppelpunkte („:“). Auf diese Weise können Sie die Quelle des Protokolleintrags identifizieren.
- Der Rust-Modulpfad, der den Code enthält, der den Protokolleintrag generiert hat.
- Der Text der Protokollnachricht.

Die Standardausgabeformate des Tracing-Moduls verwenden ANSI-Escape-Codes, um die Ausgabe einzufärben. Beachten Sie diese Escape-Sequenzen, wenn Sie die Ausgabe filtern oder durchsuchen.

Note

Was innerhalb der `sdk_invocation_id` verschachtelten Spans angezeigt wird, ist eine eindeutige ID, die vom SDK vom Client generiert wird, um die Korrelation von Protokollnachrichten zu erleichtern. Sie hat nichts mit der Anfrage-ID zu tun, die in der Antwort von einem gefunden wurde. AWS-Service

Optimieren Sie Ihre Protokollierungsstufen

Wenn Sie eine Kiste verwenden, die eine Umgebungsfiltrierung unterstützt, können Sie z. `tracing_subscriber` B. die Ausführlichkeit der Protokolle nach Modulen steuern.

Sie können für jedes Modul dieselbe Protokollierungsebene aktivieren. Im Folgenden wird die Protokollierungsebene `trace` für jedes Modul auf festgelegt:

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

Sie können die Protokollierung auf Trace-Ebene für ein bestimmtes Modul aktivieren. Im folgenden Beispiel werden auf Ebene nur Logs von `aws_smithy_runtime` eintreffend `trace`.

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

Sie können für mehrere Module eine andere Protokollebene angeben, indem Sie sie durch Kommas trennen. Im folgenden Beispiel wird das `aws_config` Modul auf `trace` Level-Logging und das `aws_smithy_runtime` Modul auf `debug` Level-Logging gesetzt.

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

In der folgenden Tabelle werden einige der Module beschrieben, die Sie zum Filtern von Protokollnachrichten verwenden können:

Präfix	Beschreibung
<code>aws_smithy_runtime</code>	Kabelprotokollierung von Anfragen und Antworten
<code>aws_config</code>	Anmeldeinformationen werden geladen
<code>aws_sigv4</code>	Signierung von Anfragen und kanonische Anfragen

Eine Möglichkeit, herauszufinden, welche Module Sie in Ihre Protokollausgabe aufnehmen müssen, besteht darin, zuerst alles zu protokollieren und dann den Crate-Namen für die benötigten Informationen in der Protokollausgabe zu suchen. Dann können Sie die Umgebungsvariable entsprechend setzen und Ihr Programm erneut ausführen.

Eine einzelne Betriebskonfiguration des Clients überschreiben

Nachdem Sie [einen Service-Client erstellt](#) haben, wird die Konfiguration unveränderlich und gilt für alle nachfolgenden Operationen. Die Konfiguration kann zu diesem Zeitpunkt zwar nicht geändert werden, sie kann jedoch für jeden Vorgang außer Kraft gesetzt werden.

Für jeden Operation Builder steht eine `customize` Methode zur Verfügung, mit der `CustomizableOperation` Sie eine einzelne Kopie der vorhandenen Konfiguration überschreiben können. Die ursprüngliche Client-Konfiguration bleibt unverändert.

Das folgende Beispiel zeigt die Erstellung eines Amazon S3 S3-Clients, der zwei Operationen aufruft, von denen der zweite überschrieben wird, um an einen anderen zu senden. AWS-Region Alle Objektaufrufe von Amazon S3 verwenden die `us-east-1` Region, außer wenn der API-Aufruf explizit überschrieben wird, um das geänderte Objekt zu verwenden. `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

Das vorherige Beispiel bezieht sich auf Amazon S3, das Konzept ist jedoch für alle Operationen dasselbe. Für bestimmte Operationen sind möglicherweise zusätzliche Methoden aktiviert `CustomizableOperation`.

Ein Beispiel für das Hinzufügen eines Interceptors, der `customize` für eine einzelne Operation verwendet wird, finden Sie unter [Interceptor nur für eine bestimmte Operation](#)

Wiederholversuche

Das AWS SDK for Rust bietet ein standardmäßiges Wiederholungsverhalten und anpassbare Konfigurationsoptionen. Ruft auf, um AWS-Services gelegentlich unerwartete Ausnahmen zurückzugeben. Bestimmte Arten von Fehlern, wie Drosselungen oder vorübergehende Fehler, können erfolgreich sein, wenn der Aufruf erneut versucht wird.

Das Wiederholungsverhalten kann global mithilfe von Umgebungsvariablen oder Einstellungen in der gemeinsam genutzten Datei konfiguriert werden. AWS config Informationen zu diesem Ansatz finden Sie unter [Verhalten bei Wiederholungsversuchen](#) im Referenzhandbuch AWS SDKs und im Tools-Referenzhandbuch. Es enthält auch detaillierte Informationen zu Implementierungen von Wiederholungsstrategien und dazu, wie Sie sich für eine Strategie entscheiden können.

Alternativ können diese Optionen auch in Ihrem Code konfiguriert werden, wie in den folgenden Abschnitten gezeigt.

Standardkonfiguration für Wiederholungsversuche

Jeder Service-Client verwendet standardmäßig die Konfiguration der `standard` Wiederholungsstrategie, die in der Struktur bereitgestellt wird. [RetryConfig](#) Standardmäßig wird ein Anruf dreimal versucht (der erste Versuch plus zwei Wiederholungen). Darüber hinaus wird jeder Wiederholungsversuch um eine kurze, zufällige Dauer verzögert, um Wiederholungsversuche zu vermeiden. Diese Konvention ist für die meisten Anwendungsfälle geeignet, kann jedoch unter bestimmten Umständen, wie z. B. bei Systemen mit hohem Durchsatz, ungeeignet sein.

Nur einige Arten von Fehlern werden von der als wiederholbar angesehen. SDKs Beispiele für Fehler, die wiederholt werden können, sind:

- Socket-Timeouts

- Drosselung auf der Serviceseite
- vorübergehende Dienstfehler wie HTTP 5XX-Antworten

Die folgenden Beispiele gelten nicht als wiederholbar:

- Fehlende oder ungültige Parameter
- Authentifizierungs-/Sicherheitsfehler
- Ausnahmen bei Fehlkonfigurationen

Sie können die standard Wiederholungsstrategie anpassen, indem Sie die maximale Anzahl an Versuchen, Verzögerungen und die Backoff-Konfiguration festlegen.

Maximale Anzahl an Versuchen

Sie können die maximale Anzahl der Versuche in Ihrem Code anpassen, indem Sie eine Änderung [RetryConfig](#) in Ihrem Code angeben `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;
```

Verzögerungen und Rückschläge

Wenn ein erneuter Versuch erforderlich ist, wartet die standardmäßige Wiederholungsstrategie, bevor sie den nächsten Versuch durchführt. Die Verzögerung beim ersten Versuch ist gering, nimmt aber bei späteren Wiederholungen exponentiell zu. Die maximale Verzögerung ist begrenzt, sodass sie nicht zu groß wird.

Zufälliger Jitter wird auf die Verzögerungen zwischen allen Versuchen angewendet. Der Jitter trägt dazu bei, die Auswirkungen großer Flotten zu mildern, die zu erneuten Stürmen führen können.

Eine eingehendere Erläuterung zu exponentiellem Backoff und Jitter finden Sie unter [Exponentieller Rückfluss und Jitter im Architektur-Blog.AWS](#)

Sie können die Verzögerungseinstellungen in Ihrem Code anpassen, indem Sie eine Änderung an Ihrem Code angeben. `RetryConfigaws_config::defaults` Der folgende Code legt die Konfiguration so fest, dass der erste Wiederholungsversuch um bis zu 100 Millisekunden verzögert wird und dass die maximale Zeitspanne zwischen jedem Wiederholungsversuch 5 Sekunden beträgt.

```
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;
```

Adaptiver Wiederholungsmodus

Als Alternative zur Moduswiederholungsstrategie ist die standard adaptive Moduswiederholungsstrategie ein fortschrittlicher Ansatz, bei dem nach der idealen Anforderungsrate gesucht wird, um Drosselungsfehler zu minimieren.

Note

Adaptive Wiederholungen sind ein erweiterter Wiederholungsmodus. Die Verwendung dieser Strategie wird in der Regel nicht empfohlen. Weitere Informationen finden Sie unter [Verhalten bei Wiederholungsversuchen](#) im Referenzhandbuch AWS SDKs und im Tools-Referenzhandbuch.

Adaptive Wiederholungen umfassen alle Funktionen von Standardwiederholungen. Es fügt einen clientseitigen Ratenbegrenzer hinzu, der die Rate gedrosselter Anfragen im Vergleich zu Anfragen ohne Drosselung misst. Außerdem wird der Datenverkehr so begrenzt, dass versucht wird, innerhalb einer sicheren Bandbreite zu bleiben, sodass im Idealfall keine Drosselungsfehler auftreten.

Die Rate passt sich in Echtzeit an sich ändernde Betriebsbedingungen und Verkehrsmuster an und kann die Verkehrsrate entsprechend erhöhen oder verringern. Entscheidend ist, dass der Ratenbegrenzer erste Versuche in Szenarien mit hohem Verkehrsaufkommen verzögern kann.

Sie können die adaptive Wiederholungsstrategie im Code auswählen, indem Sie Folgendes angeben: [RetryConfig](#)

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

Timeouts

Das AWS SDK for Rust bietet verschiedene Einstellungen für die Verwaltung von Anforderungs-Timeouts und blockierten Datenströmen. Diese helfen Ihrer Anwendung, sich optimal zu verhalten, wenn unerwartete Verzögerungen und Ausfälle im Netzwerk auftreten.

API-Timeouts

Wenn es vorübergehende Probleme gibt, die dazu führen können, dass Anforderungsversuche lange dauern oder ganz fehlschlagen, ist es wichtig, Timeouts zu überprüfen und festzulegen, damit Ihre Anwendung schnell fehlschlägt und sich optimal verhält. Anfragen, die fehlschlagen, können vom SDK automatisch wiederholt werden. Es empfiehlt sich, Timeouts sowohl für den einzelnen Versuch als auch für die gesamte Anfrage festzulegen.

Das SDK für Rust bietet ein Standard-Timeout für den Verbindungsaufbau für eine Anfrage. Für das SDK ist standardmäßig keine maximale Wartezeit für den Empfang einer Antwort auf einen Anforderungsversuch oder für die gesamte Anfrage festgelegt. Die folgenden Timeout-Optionen sind verfügbar:

Parameter	Standardwert	Beschreibung
Verbindungs-Timeout	3,1 Sekunden	Die maximale Wartezeit, bis eine Verbindung hergestellt ist, bevor der Vorgang abgebrochen wird.

Parameter	Standardwert	Beschreibung
Timeout für den Vorgang	Keine	Die maximale Wartezeit, bis eine Antwort vom SDK für Rust eingeht, einschließlich aller Wiederholungsversuche.
Timeout für den Versuch eines Vorgangs	Keine	Die maximale Wartezeit auf einen einzelnen HTTP-Versuch. Danach kann der API-Aufruf erneut versucht werden.
Timeout beim Lesen	Keine	Die maximale Wartezeit bis zum Lesen des ersten Bytes einer Antwort ab dem Zeitpunkt, an dem die Anforderung initiiert wird.

Das folgende Beispiel zeigt die Konfiguration eines Amazon S3 S3-Clients mit benutzerdefinierten Timeout-Werten:

```
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);
```

Wenn Sie sowohl Vorgangs- als auch Versuchs-Timeouts zusammen verwenden, legen Sie ein festes Limit für die Gesamtzeit fest, die für alle Versuche bei Wiederholungen aufgewendet wird. Sie legen außerdem fest, dass eine einzelne HTTP-Anfrage bei einer langsamen Anfrage schnell fehlschlägt.

Als Alternative zum Festlegen dieser Timeout-Werte auf dem Service-Client für alle Operationen können Sie [sie für eine einzelne Anfrage konfigurieren oder überschreiben](#).

⚠ Important

Timeouts für Vorgänge und Versuche gelten nicht für Streaming-Daten, die verbraucht werden, nachdem das SDK für Rust eine Antwort zurückgegeben hat. Ein Beispiel: Wenn Daten von einem `ByteStream` Mitglied einer Antwort abgerufen werden, gibt es keine Zeitüberschreitungen bei Vorgängen.

Der Stream-Schutz ist ins Stocken geraten

Das SDK für Rust bietet eine weitere Form von Timeout im Zusammenhang mit der Erkennung blockierter Streams. Ein blockierter Stream ist ein Upload- oder Download-Stream, der länger als eine konfigurierte Kulanzzeit keine Daten erzeugt. Dadurch wird verhindert, dass Anwendungen auf unbestimmte Zeit hängen bleiben und niemals Fortschritte machen.

Der Schutz vor blockierten Streams gibt einen Fehler zurück, wenn ein Stream länger als den akzeptablen Zeitraum inaktiv ist.

Standardmäßig aktiviert das SDK für Rust den Schutz vor blockierten Streams sowohl für Uploads als auch für Downloads und sucht nach einer Aktivität von mindestens 1 Byte/Sekunde mit einer großzügigen Nachfrist von 20 Sekunden.

Das folgende Beispiel zeigt eine benutzerdefinierte Version, die den Upload-Schutz deaktiviert und die Übergangszeit für den Fall, `StalledStreamProtectionConfig` dass keine Aktivität stattfindet, auf 10 Sekunden ändert:

```
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

Der Schutz vor blockierten Streams ist eine erweiterte Konfigurationsoption. Wir empfehlen, diese Werte nur zu ändern, wenn Ihre Anwendung eine höhere Leistung benötigt oder wenn sie ein anderes Problem verursacht.

Deaktivieren Sie den Schutz vor blockierten Streams

Das folgende Beispiel zeigt, wie der Schutz vor blockierten Streams vollständig deaktiviert werden kann:

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

Benutze die AWS SDK for Rust

Lernen Sie gängige und empfohlene Methoden zur Verwendung von AWS SDK for Rust für die Arbeit mit AWS Diensten kennen.

Themen

- [Anfragen stellen](#)
- [Bewährte Methoden](#)
- [Nebenläufigkeit](#)
- [Fehlerbehandlung](#)
- [Lambda-Funktionen erstellen](#)
- [Paginierung](#)
- [Vorsigniert erstellen URLs](#)
- [Komponententests](#)
- [Waiter](#)

Anfragen stellen

Um eine Anfrage an zu stellen AWS-Service, müssen Sie zunächst [einen Service-Client erstellen](#). Für jeden Code, den AWS-Service Sie verwenden, gibt es eine eigene Kiste und eine eigene `Client` für die Interaktion mit ihm.

Das `Client` macht eine Methode für jeden API-Vorgang verfügbar, der vom Dienst verfügbar gemacht wird. Der Rückgabewert jeder dieser Methoden ist ein „Fluent Builder“, bei dem verschiedene Eingaben für diese API durch Verkettung von Funktionsaufrufen im Builder-Stil hinzugefügt werden. Rufen Sie nach dem Aufrufen der Methoden des Dienstes auf, `send()` um eine zu erhalten [Future](#), die entweder zu einer erfolgreichen Ausgabe oder zu einer führt. `SdkError` Weitere Informationen zu `SdkError` finden Sie unter [Fehlerbehandlung](#).

Das folgende Beispiel zeigt eine grundlegende Operation mit Amazon S3 zur Erstellung eines Buckets in `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;
```

Jede Service-Kiste enthält zusätzliche Module, die für API-Eingaben verwendet werden, wie zum Beispiel die folgenden:

- Das `types` Modul verfügt über Strukturen oder Aufzählungen, um komplexere strukturierte Informationen bereitzustellen.
- Das `primitives` Modul verfügt über einfachere Typen zur Darstellung von Daten wie Datums- und Uhrzeitangaben oder binären Blobs.

Eine detailliertere Organisation und Informationen zu Crate finden Sie in der [API-Referenzdokumentation](#) für den Service Crate. Zum Beispiel hat die `aws-sdk-s3` Kiste für den Amazon Simple Storage Service mehrere [Module](#). Zwei davon sind:

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

Bewährte Methoden

Im Folgenden finden Sie bewährte Methoden für die Verwendung von AWS SDK for Rust

Verwenden Sie SDK-Clients nach Möglichkeit wieder

Je nachdem, wie ein SDK-Client aufgebaut ist, kann die Erstellung eines neuen Clients dazu führen, dass jeder Client seine eigenen HTTP-Verbindungspools, Identitätscaches usw. beibehält. Wir empfehlen, einen Client gemeinsam zu nutzen oder zumindest gemeinsam `SdkConfig` zu nutzen,

um den Aufwand einer teuren Ressourcenerstellung zu vermeiden. Alle SDK-Clients implementieren `Clone` die Aktualisierung in Form eines einzigen atomaren Referenzzählers.

API-Timeouts konfigurieren

Das SDK bietet Standardwerte für einige Timeout-Optionen wie Verbindungs-Timeout und Socket-Timeouts, jedoch nicht für API-Aufruf-Timeouts oder einzelne API-Aufrufversuche. Es empfiehlt sich, Timeouts sowohl für den einzelnen Versuch als auch für die gesamte Anfrage festzulegen. Auf diese Weise wird sichergestellt, dass Ihre Anwendung schnell und optimal fehlschlägt, wenn vorübergehende Probleme auftreten, die dazu führen können, dass Anforderungsversuche länger dauern können, oder bei schwerwiegenden Netzwerkproblemen.

Weitere Informationen zur Konfiguration von Betriebs-Timeouts finden Sie unter [Timeouts](#)

Nebenläufigkeit

Das bietet AWS SDK for Rust keine Parallelitätssteuerung, aber Benutzer haben viele Möglichkeiten, ihre eigenen zu implementieren.

Bedingungen

Begriffe, die sich auf dieses Thema beziehen, sind leicht zu verwechseln, und einige Begriffe sind zu Synonymen geworden, obwohl sie ursprünglich unterschiedliche Konzepte darstellten. In diesem Leitfaden werden wir Folgendes definieren:

- **Aufgabe:** Eine „Arbeitseinheit“, die Ihr Programm bis zum Abschluss ausführen wird oder versucht, es bis zum Abschluss auszuführen.
- **Sequentielles Rechnen:** Wenn mehrere Aufgaben nacheinander ausgeführt werden.
- **Gleichzeitiges Rechnen:** Wenn mehrere Aufgaben in überlappenden Zeiträumen ausgeführt werden.
- **Parallelität:** Die Fähigkeit eines Computers, mehrere Aufgaben in beliebiger Reihenfolge auszuführen.
- **Multitasking:** Die Fähigkeit eines Computers, mehrere Aufgaben gleichzeitig auszuführen.
- **Race Condition:** Wenn sich das Verhalten Ihres Programms ändert, je nachdem, wann eine Aufgabe gestartet wird oder wie lange es dauert, eine Aufgabe zu bearbeiten.
- **Streit:** Konflikt über den Zugriff auf eine gemeinsam genutzte Ressource. Wenn zwei oder mehr Aufgaben gleichzeitig auf eine Ressource zugreifen wollen, ist diese Ressource „umstritten“.

- **Deadlock:** Ein Zustand, in dem keine Fortschritte mehr erzielt werden können. Dies ist in der Regel der Fall, weil zwei Aufgaben die Ressourcen des jeweils anderen übernehmen wollen, aber keine Aufgabe ihre Ressourcen freigibt, bis die Ressource des anderen wieder verfügbar ist. Deadlocks führen dazu, dass ein Programm ganz oder teilweise nicht mehr reagiert.

Ein einfaches Beispiel

Unser erstes Beispiel ist ein sequentielles Programm. In späteren Beispielen werden wir diesen Code mithilfe von Parallelitätstechniken ändern. In `main()` späteren Beispielen wird dieselbe `build_client_and_list_objects_to_download()` Methode wiederverwendet und Änderungen daran vorgenommen.

Die folgende Beispielaufgabe besteht darin, alle Dateien in einem Amazon Simple Storage Service-Bucket herunterzuladen:

1. Beginnen Sie mit der Auflistung aller Dateien. Speichern Sie die Schlüssel in einer Liste.
2. Iterieren Sie die Liste und laden Sie nacheinander jede Datei herunter

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

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

    (client, objects_to_download)
}
```

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

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

Note

In diesen Beispielen behandeln wir keine Fehler und gehen davon aus, dass der Beispiel-Bucket keine Objekte mit Schlüsseln enthält, die wie Dateipfade aussehen. Daher werden wir uns nicht mit der Erstellung verschachtelter Verzeichnisse befassen.

Aufgrund der Architektur moderner Computer können wir dieses Programm so umschreiben, dass es viel effizienter ist. Wir werden das in einem späteren Beispiel tun, aber zuerst wollen wir ein paar weitere Konzepte lernen.

Eigentum und Wandelbarkeit

Jeder Wert in Rust hat einen einzigen Besitzer. Wenn ein Besitzer den Gültigkeitsbereich verlässt, werden auch alle Werte, die er besitzt, gelöscht. Der Eigentümer kann entweder einen oder mehrere unveränderliche Verweise auf einen Wert oder eine einzelne veränderbare Referenz angeben. Der Rust-Compiler ist dafür verantwortlich, dass keine Referenz ihren Besitzer überlebt.

Zusätzliche Planung und Gestaltung sind erforderlich, wenn mehrere Aufgaben wechselseitig auf dieselbe Ressource zugreifen müssen. Beim sequentiellen Rechnen kann jede Aufgabe ohne Konflikte wechselseitig auf dieselbe Ressource zugreifen, da sie nacheinander in einer Reihenfolge

ausgeführt werden. Beim gleichzeitigen Rechnen können Aufgaben jedoch in beliebiger Reihenfolge und gleichzeitig ausgeführt werden. Daher müssen wir mehr tun, um dem Compiler zu beweisen, dass mehrere veränderbare Referenzen unmöglich sind (oder zumindest abstürzen, falls sie auftreten).

Die Rust-Standardbibliothek bietet viele Tools, die uns dabei helfen. Weitere Informationen zu diesen Themen finden Sie im Buch [Variablen und Veränderlichkeit und Grundlegendes zur Eigentümerschaft](#) in der Programmiersprache Rust.

Weitere Begriffe!

Im Folgenden finden Sie Listen von „Synchronisationsobjekten“. Insgesamt sind sie die Werkzeuge, die notwendig sind, um den Compiler davon zu überzeugen, dass unser Parallelprogramm nicht gegen Eigentumsregeln verstößt.

[Standard-Synchronisationsobjekte für Bibliotheken:](#)

- [Bogen](#): Ein atomisch R-Referenz-C-montierter A-Zeiger. Wenn Daten in einer Box verpackt sind `Arc`, können sie frei geteilt werden, ohne sich Sorgen machen zu müssen, dass ein bestimmter Eigentümer den Wert vorzeitig verliert. In diesem Sinne wird das Eigentum an dem Wert „geteilt“. Werte innerhalb eines `Arc` können nicht veränderbar sein, können aber innerlich [veränderbar](#) sein.
- [Barriere](#): Stellt sicher, dass mehrere Threads darauf warten, dass sie jeweils einen Punkt im Programm erreichen, bevor sie die Ausführung gemeinsam fortsetzen.
- [Condvar](#): Eine Bedingungsvariable, die die Möglichkeit bietet, einen Thread zu blockieren, während auf das Eintreten eines Ereignisses gewartet wird.
- [Mutex](#): Ein Mutual Exclusion-Mechanismus, der sicherstellt, dass jeweils höchstens ein Thread auf einige Daten zugreifen kann. Im Allgemeinen sollte eine Mutex Sperre niemals an einer `.await` Stelle im Code aktiviert werden.

[Synchronisationsobjekte für Tokio:](#)

Sie AWS SDKs sollen zwar `async -runtime-agnostisch` sein, wir empfehlen jedoch die Verwendung von Synchronisationsobjekten für bestimmte Fälle. `tokio`

- [Mutex](#): Ähnlich wie die Standardbibliothek `Mutex`, aber mit etwas höheren Kosten. Im Gegensatz zum Standard `Mutex` kann dieser an einem `.await` Punkt im Code gespeichert werden.
- [Semaphore](#): Eine Variable, die verwendet wird, um den Zugriff mehrerer Aufgaben auf eine gemeinsame Ressource zu steuern.

Wir haben unser Beispiel umgeschrieben, um es effizienter zu machen (Single-Thread-Parallelität)

Im folgenden modifizierten Beispiel verwenden wir die Methode, ALLE

[futures_util::future::join_all](#) `get_object` Anfragen gleichzeitig auszuführen.

```
#[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;
}
```

Dies ist der einfachste Weg, um von Parallelität zu profitieren, hat aber auch einige Probleme, die auf den ersten Blick vielleicht nicht offensichtlich sind:

1. Wir erstellen alle Anforderungseingaben gleichzeitig. Wenn wir nicht genug Speicher haben, um alle `get_object` Anforderungseingaben aufzunehmen, tritt ein Zuweisungsfehler `out-of-memory` auf.
2. Wir erschaffen alle Zukünfte und erwarten sie gleichzeitig. Amazon S3 drosselt Anfragen, wenn wir versuchen, zu viel auf einmal herunterzuladen.

Um diese beiden Probleme zu beheben, müssen wir die Anzahl der Anfragen, die wir gleichzeitig senden, begrenzen. Wir machen das mit einer 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;
}
```

Wir haben das potenzielle Problem mit der Speichernutzung behoben, indem wir die Anforderungserstellung in den `async` Block verschoben haben. Auf diese Weise werden Anfragen erst erstellt, wenn es Zeit ist, sie zu senden.

Note

Wenn Sie den Speicher dafür haben, ist es möglicherweise effizienter, alle Ihre Anforderungseingaben auf einmal zu erstellen und sie im Speicher zu behalten, bis sie zum Senden bereit sind. Um dies zu versuchen, verschieben Sie die Erstellung von Anforderungseingaben außerhalb des `async` Blocks.

Wir haben auch das Problem behoben, dass zu viele Anfragen gleichzeitig gesendet wurden, indem wir die Anzahl der Anfragen im Flug auf `beschränkt habenCONCURRENCY_LIMIT`.

Note

Der richtige Wert für `CONCURRENCY_LIMIT` ist für jedes Projekt anders. Wenn Sie Ihre eigenen Anfragen erstellen und senden, versuchen Sie, ihn so hoch wie möglich zu setzen, ohne dass Drosselungsfehler auftreten. Es ist zwar möglich, Ihr Parallelitätslimit auf der Grundlage des Verhältnisses zwischen erfolgreichen und gedrosselten Antworten, die ein Dienst zurücksendet, dynamisch zu aktualisieren, aber das ist aufgrund seiner Komplexität nicht Gegenstand dieses Handbuchs.

Wir haben unser Beispiel umgeschrieben, um es effizienter zu machen (Parallelität mit mehreren Threads)

In den beiden vorherigen Beispielen haben wir unsere Anfragen gleichzeitig ausgeführt. Das ist zwar effizienter, als sie synchron auszuführen, aber wir können die Dinge noch effizienter gestalten, indem wir Multithreading verwenden. Dazu müssen wir sie als separate Aufgaben erstellen. `tokio`

Note

Für dieses Beispiel müssen Sie die Multithread-Laufzeit `tokio` verwenden. Diese Laufzeit befindet sich hinter der `rt-multi-thread` Funktion. Und natürlich müssen Sie Ihr Programm auf einem Multicore-Computer ausführen.

```
// 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;
}
```

Die Aufteilung der Arbeit in Aufgaben kann komplex sein. I/O (Eingabe/Ausgabe) blockiert normalerweise. Bei Laufzeiten kann es schwierig sein, die Anforderungen von Aufgaben mit langer Laufzeit und denen von Aufgaben mit kurzer Laufzeit in Einklang zu bringen. Für welche Laufzeit Sie sich auch entscheiden, lesen Sie unbedingt deren Empfehlungen, wie Sie Ihre Arbeit am effizientesten in Aufgaben unterteilen können. Empfehlungen zur tokio Laufzeit finden Sie unter [Modul tokio::task](#).

Debuggen von Apps mit mehreren Threads

Aufgaben, die gleichzeitig ausgeführt werden, können in beliebiger Reihenfolge ausgeführt werden. Daher können die Protokolle gleichzeitiger Programme sehr schwer zu lesen sein. Im SDK für Rust empfehlen wir die Verwendung des `tracing` Logging-Systems. Es kann Logs nach ihren spezifischen Aufgaben gruppieren, unabhängig davon, wann sie ausgeführt werden. Anleitungen finden Sie unter [Protokollierung von AWS SDK for Rust Code aktivieren](#).

Ein sehr nützliches Tool zur Identifizierung von Aufgaben, die gesperrt sind [tokio-console](#), ist ein Diagnose- und Debugging-Tool für asynchrone Rust-Programme. Indem Sie Ihr Programm instrumentieren und ausführen und dann die `tokio-console` App ausführen, können Sie eine Live-Ansicht der Aufgaben sehen, die Ihr Programm gerade ausführt. Diese Ansicht enthält hilfreiche Informationen wie die Zeit, die eine Aufgabe damit verbracht hat, auf gemeinsam genutzte Ressourcen zu warten, oder wie oft sie abgefragt wurde.

Fehlerbehandlung

Es ist wichtig zu verstehen, wie und wann die Fehler AWS SDK for Rust zurückgegeben werden, um qualitativ hochwertige Anwendungen mithilfe des SDK zu erstellen. In den folgenden Abschnitten werden die verschiedenen Fehler beschrieben, auf die Sie beim SDK stoßen können, und wie Sie sie angemessen behandeln können.

Bei jeder Operation wird ein `Result` Typ zurückgegeben, bei dem der Fehlertyp auf gesetzt ist [SdkError<E, R = HttpResponse>](#). `SdkError` ist eine Aufzählung mit mehreren möglichen Typen, die als Varianten bezeichnet werden.

Dienstfehler

Die häufigste Art von Fehler ist [SdkError::ServiceError](#). Dieser Fehler stellt eine Fehlerantwort von einem der AWS-Service. Wenn Sie beispielsweise versuchen, ein Objekt von Amazon S3 abzurufen, das nicht existiert, gibt Amazon S3 eine Fehlerantwort zurück.

Wenn Sie auf eine stoßen, bedeutet `SdkError::ServiceError` dies, dass Ihre Anfrage erfolgreich an die gesendet wurde AWS-Service, aber nicht bearbeitet werden konnte. Dies kann an Fehlern in den Parametern der Anforderung oder an Probleme auf Seiten des Services liegen.

Die Details zur Fehlerantwort sind in der Fehlervariante enthalten. Das folgende Beispiel zeigt, wie Sie bequem zur zugrunde liegenden `ServiceError` Variante gelangen und verschiedene Fehlerfälle behandeln können:

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

Fehler-Metadaten

Jeder Servicefehler hat zusätzliche Metadaten, auf die durch den Import dienstspezifischer Merkmale zugegriffen werden kann.

- Das `<service>::error::ProvideErrorMetadata` Merkmal bietet Zugriff auf alle verfügbaren zugrunde liegenden Rohfehlercodes und Fehlermeldungen, die vom Dienst zurückgegeben werden.
- Für Amazon S3 ist dieses Merkmal [aws_sdk_s3::error::ProvideErrorMetadata](#).

Sie können auch Informationen abrufen, die bei der Behebung von Servicefehlern nützlich sein können:

- Das `<service>::operation::RequestId` Merkmal fügt Erweiterungsmethoden hinzu, um die eindeutige AWS Anforderungs-ID abzurufen, die vom Dienst generiert wurde.

- Für Amazon S3 ist dieses Merkmal [aws_sdk_s3::operation::RequestId](#).
- Das `<service>::operation::RequestIdExt` Merkmal fügt die `extended_request_id()` Methode hinzu, um eine zusätzliche, erweiterte Anfrage-ID zu erhalten.
 - Wird nur von einigen Diensten unterstützt.
- Für Amazon S3 ist dieses Merkmal [aws_sdk_s3::operation::RequestIdExt](#).

Detaillierter Fehler beim Drucken mit **DisplayErrorContext**

Fehler im SDK sind im Allgemeinen das Ergebnis einer Reihe von Fehlern wie:

1. Das Versenden einer Anfrage ist fehlgeschlagen, weil der Connector einen Fehler zurückgegeben hat.
2. Der Connector hat einen Fehler zurückgegeben, weil der Anbieter für Anmeldeinformationen einen Fehler zurückgegeben hat.
3. Der Anbieter für Anmeldeinformationen hat einen Fehler zurückgegeben, weil er einen Dienst aufgerufen hat und dieser Dienst einen Fehler zurückgegeben hat.
4. Der Dienst hat einen Fehler zurückgegeben, weil die Anforderung der Anmeldeinformationen nicht die richtige Autorisierung hatte.

Standardmäßig wird bei der Anzeige dieses Fehlers nur die Meldung „Versandfehler“ ausgegeben. Hier fehlen Details, die zur Behebung des Fehlers beitragen könnten. Das SDK für Rust bietet einen einfachen Fehlerreporter namens `DisplayErrorContext`.

- Die `<service>::error::DisplayErrorContext` Struktur fügt Funktionen hinzu, um den vollständigen Fehlerkontext auszugeben.
 - Für Amazon S3 ist diese Struktur [aws_sdk_s3::error::DisplayErrorContext](#).

Wenn wir den anzuzeigenden Fehler einpacken und ausdrucken, `DisplayErrorContext` wird eine viel detailliertere Meldung angezeigt, die der folgenden ähnelt:

```
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 SS0") },
                    "x-amzn-requestid": HeaderValue
{ _private: H0("1b6d7476-f5ec-4a16-9890-7684ccee7d01") }
                  }
                },
                body: SdkBody {
                  inner: Once(
                    Some(
                      b"{

```

```

        "message": "Session token not
found or invalid",
        "__type":
"com.amazonaws.switchboard.portal#UnauthorizedException"}"
    )
    ),
    retryable: true
},
extensions: Extensions {
    extensions_02x: Extensions,
    extensions_1x: Extensions
}
}
)
}
),
connection: Unknown
}
}
)

```

Lambda-Funktionen erstellen

Eine ausführliche Dokumentation zur Entwicklung von AWS Lambda Funktionen mit dem AWS SDK for Rust finden Sie unter [Erstellen von Lambda-Funktionen mit Rust](#) im AWS Lambda Entwicklerhandbuch. Diese Dokumentation führt Sie durch die Verwendung von:

- Die Rust Lambda Runtime-Client-Crate für Kernfunktionen, [aws-lambda-rust-runtime](#)
- [Das empfohlene Befehlszeilentool für die Bereitstellung der Rust-Funktionsbinärdatei auf Lambda mit Cargo Lambda.](#)

Zusätzlich zu den angeleiteten Beispielen, die im AWS Lambda Developer Guide enthalten sind, gibt es auch Beispiele für einen Lambda-Rechner, die im [AWS SDK Code Examples Repository](#) unter GitHub verfügbar sind.

Paginierung

Viele AWS Operationen geben gekürzte Ergebnisse zurück, wenn die Nutzlast zu groß ist, um sie in einer einzigen Antwort zurückzugeben. Stattdessen gibt der Dienst einen Teil der Daten und ein Token zurück, um den nächsten Satz von Elementen abzurufen. Dieses Muster wird als Paginierung bezeichnet.

AWS SDK for Rust Dazu gehören Erweiterungsmethoden `into_paginator` für Operation Builder, mit denen Sie die Ergebnisse automatisch paginieren können. Sie müssen nur den Code schreiben, der die Ergebnisse verarbeitet. Allen Generatoren für Paginierungsoperationen steht eine `into_paginator()` Methode zur Verfügung, mit der die Ergebnisse paginiert werden [PaginationStream<Item>](#) können.

- In Amazon S3 ist ein Beispiel dafür

[aws_sdk_s3::operation::list_objects_v2::builders::ListObjectsV2FluentBuilder:::](#)

In den folgenden Beispielen wird Amazon Simple Storage Service verwendet. Die Konzepte sind jedoch für jeden Service identisch, der über einen oder mehrere Seiten verfügt. APIs

Das folgende Codebeispiel zeigt das einfachste Beispiel, in dem die [try_collect\(\)](#) Methode verwendet wird, um alle paginierten Ergebnisse in einem zusammenzufassen: `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<_>>();
```

Manchmal möchten Sie mehr Kontrolle über das Paging haben und nicht alles auf einmal in den Speicher ziehen. Im folgenden Beispiel wird über Objekte in einem Amazon S3 S3-Bucket iteriert, bis keine weiteren mehr vorhanden sind.

```
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);
    }
}
```

Vorsigniert erstellen URLs

Sie können Anfragen für einige AWS API-Operationen vorab signieren, sodass ein anderer Aufrufer die Anfrage später verwenden kann, ohne seine eigenen Anmeldeinformationen angeben zu müssen.

Nehmen wir zum Beispiel an, dass Jane Zugriff auf ein Amazon Simple Storage Service (Amazon S3) -Objekt hat und den Objektzugriff vorübergehend mit Alejandro teilen möchte. Jane kann eine vorab signierte `GetObject` Anfrage zur Weitergabe an Alejandro generieren, sodass er das Objekt herunterladen kann, ohne Zugriff auf Janes Anmeldeinformationen zu benötigen oder eigene zu haben. Die von der vorsignierten URL verwendeten Anmeldeinformationen gehören Jane, da sie die AWS Benutzerin ist, die die URL generiert hat.

Weitere Informationen zu presigned URLs in Amazon S3 finden Sie unter [Working with presigned URLs](#) im Amazon Simple Storage Service-Benutzerhandbuch.

Grundlagen der Vorsignierung

Das AWS SDK for Rust stellt eine `presigned()` Methode zur Operation Fluent-Builder bereit, die verwendet werden kann, um eine vorab signierte Anfrage abzurufen.

Das folgende Beispiel erstellt eine vorsignierte `GetObject` Anfrage für Amazon S3. Die Anfrage ist nach ihrer Erstellung 5 Minuten lang gültig.

```
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?;
```

Die `presigned()` Methode gibt a zurück `Result<PresignedRequest, SdkError<E, R>>`.

Die zurückgegebene `PresignedRequest` Datei enthält Methoden zum Abrufen der Komponenten einer HTTP-Anfrage, einschließlich der Methode, des URI und aller Header. All diese müssen an den Dienst gesendet werden, falls vorhanden, damit die Anfrage gültig ist. Viele vorab signierte Anfragen können jedoch allein durch die URI dargestellt werden.

Vorsignierung **POST** und Anfragen **PUT**

Viele Operationen, die vorsignierbar sind, benötigen nur eine URL und müssen als HTTP-Anfragen gesendet werden. GET Einige Operationen benötigen jedoch einen Hauptteil und müssen in einigen Fällen als HTTP POST - oder PUT HTTP-Anfrage zusammen mit Headern gesendet werden. Das Vorsignieren dieser Anfragen ist identisch mit dem Vorsignieren von GET Anfragen, das Aufrufen der vorab signierten Anfrage ist jedoch komplizierter.

Im Folgenden finden Sie ein Beispiel für die Vorsignierung einer Amazon S3 PutObject S3-Anfrage und [http::request::Request](#) deren Konvertierung in eine, die mit einem HTTP-Client Ihrer Wahl gesendet werden kann.

Um diese `into_http_1x_request()` Methode zu verwenden, fügen Sie die `http-1x` Funktion zu Ihrer `aws-sdk-s3` Kiste in Ihrer `Cargo.toml` Datei hinzu:

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

Quelldatei:

```
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);
```

Eigenständiger Unterzeichner

Note

Dies ist ein Anwendungsfall für Fortgeschrittene. Es wird für die meisten Benutzer nicht benötigt oder empfohlen.

Es gibt einige Anwendungsfälle, in denen es erforderlich ist, eine signierte Anfrage außerhalb des SDK für Rust-Kontextes zu erstellen. Dafür können Sie die [aws-sigv4](#) Kiste unabhängig vom SDK verwenden.

Das Folgende ist ein Beispiel zur Veranschaulichung der grundlegenden Elemente. Weitere Informationen finden Sie in der Crate-Dokumentation.

Fügen Sie die `http` Kisten `aws-sigv4` und zu Ihrer `Cargo.toml` Datei hinzu:

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

Quelldatei:

```
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);
```

Komponententests

Es gibt zwar viele Möglichkeiten, Unit-Tests in Ihrem AWS SDK for Rust Projekt zu implementieren, aber es gibt einige, die wir empfehlen:

- Verwenden Sie es [automock](#) aus der [mockall](#) Kiste, um Ihre Tests zu erstellen und auszuführen.
- Verwenden Sie die AWS Smithy-Laufzeitumgebung [StaticReplayClient](#), um einen gefälschten HTTP-Client zu erstellen, der anstelle des Standard-HTTP-Clients verwendet werden kann, der normalerweise verwendet wird. Dieser Client gibt die von Ihnen angegebenen HTTP-Antworten zurück, anstatt mit dem Dienst über das Netzwerk zu kommunizieren, sodass Tests bekannte Daten für Testzwecke abrufen.

Generieren Sie automatisch Mocks mithilfe von Mockall

Sie können die meisten Mock-Implementierungen, die Ihre Tests benötigen, automatisch generieren, indem Sie die beliebte [automock](#) Version aus der Kiste verwenden. [mockall](#)

In diesem Beispiel wird eine benutzerdefinierte Methode namens `getestet`.

`determine_prefix_file_size()` Diese Methode ruft eine benutzerdefinierte `list_objects()` Wrapper-Methode auf, die Amazon S3 aufruft. Durch Spott kann die `determine_prefix_file_size()` Methode getestet werden `list_objects()`, ohne Amazon S3 tatsächlich zu kontaktieren.

1. Fügen Sie in einer Befehlszeile für Ihr Projektverzeichnis die [mockall](#) Kiste als Abhängigkeit hinzu:

```
$ cargo add mockall
```

Dadurch wird die Kiste dem `[dependencies]` Abschnitt Ihrer `Cargo.toml` Datei hinzugefügt.

2. Fügen Sie das `automock` Modul aus der `mockall` Kiste hinzu.

Schließen Sie auch alle anderen Bibliotheken ein AWS-Service, die sich auf das beziehen, das Sie testen, in diesem Fall Amazon S3.

```
use aws_sdk_s3 as s3;
```

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

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

- Fügen Sie als Nächstes Code hinzu, der bestimmt, welche von zwei Implementierungen der Amazon S3 S3-Wrapper-Struktur der Anwendung verwendet werden soll.
 - Der echte, der für den Zugriff auf Amazon S3 über das Netzwerk geschrieben wurde.
 - Die Scheinimplementierung, generiert von `mockall`.

In diesem Beispiel erhält die ausgewählte Datei den Namen `S3`. Die Auswahl ist abhängig von dem `test` Attribut:

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

- Die `S3Impl` Struktur ist die Implementierung der Amazon S3 S3-Wrapper-Struktur, an die tatsächlich Anfragen gesendet AWS werden.
 - Wenn das Testen aktiviert ist, wird dieser Code nicht verwendet, da die Anfrage an den Mock gesendet wird und nicht AWS. Das `dead_code` Attribut weist den Linter an, kein Problem zu melden, wenn der `S3Impl` Typ nicht verwendet wird.
 - Die Bedingung `#[cfg_attr(test, automock)]` gibt an, dass das `automock` Attribut gesetzt werden sollte, wenn das Testen aktiviert ist. Dies weist darauf `mockall` hin, `S3Impl` dass ein Modell generiert werden soll, das benannt wird `MockS3Impl`.
 - In diesem Beispiel ist die `list_objects()` Methode der Aufruf, den Sie verspotten möchten. `automock` erstellt automatisch eine `expect_list_objects()` Methode für Sie.

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

#[cfg_attr(test, automock)]
impl S3Impl {
    #[allow(dead_code)]
```

```

pub fn new(inner: s3::Client) -> Self {
    Self { inner }
}

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

```

5. Erstellen Sie die Testfunktionen in einem Modul mit dem Namen `test`.

- Die Bedingung `#[cfg(test)]` gibt an, dass das Testmodul erstellt werden `mockall` soll, wenn das `test true` Attribut

```

#[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);
}
}

```

- Jeder Test verwendet `let mut mock = MockS3Impl::default();`, um eine mock Instanz von `MockS3Impl` zu erstellen.
 - Es verwendet die `expect_list_objects()` Methode des Mocks (die automatisch von `MockS3Impl` erstellt wurde), um das erwartete Ergebnis festzulegen, wenn die `list_objects()` Methode an anderer Stelle im Code verwendet wird.
 - Nachdem die Erwartungen festgelegt wurden, werden diese verwendet, um die Funktion durch Aufrufen von `test_determine_prefix_file_size()` zu testen. Der zurückgegebene Wert wird anhand einer Assertion überprüft, um zu bestätigen, dass er korrekt ist.
6. Die `determine_prefix_file_size()` Funktion verwendet den Amazon S3 S3-Wrapper, um die Größe der Präfixdatei abzurufen:

```

#[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)
}
```

Der Typ `S3` wird verwendet, um das verpackte SDK für Rust-Funktionen aufzurufen, um beide Funktionen zu unterstützen, `S3Impl` und `MockS3Impl` wenn HTTP-Anfragen gestellt werden. Der automatisch generierte Mock `mockall` meldet alle Testfehler, wenn das Testen aktiviert ist.

[Den vollständigen Code für diese Beispiele finden](#) Sie unter GitHub.

Simulieren Sie den HTTP-Verkehr mithilfe von statischer Wiedergabe

Die `aws-smithy-runtime` Kiste enthält eine Testdienstprogrammklasse namens.

[StaticReplayClient](#) Diese HTTP-Client-Klasse kann anstelle des Standard-HTTP-Clients angegeben werden, wenn ein AWS-Service Objekt erstellt wird.

Bei der `StaticReplayClient` Initialisierung von geben Sie eine Liste von HTTP-Anforderungs- und Antwortpaaren als `ReplayEvent` Objekte an. Während der Test ausgeführt wird, wird jede HTTP-Anfrage aufgezeichnet und der Client gibt die nächste HTTP-Antwort, die in der nächsten in der Ereignisliste gefunden wurde, als Antwort des HTTP-Clients zurück. `ReplayEvent` Dadurch kann der Test mit bekannten Daten und ohne Netzwerkverbindung ausgeführt werden.

Statische Wiedergabe wird verwendet

Um Static Replay zu verwenden, müssen Sie keinen Wrapper verwenden. Ermitteln Sie stattdessen, wie der tatsächliche Netzwerkverkehr für die Daten aussehen sollte, die Ihr Test verwenden wird, und stellen Sie diese Verkehrsdaten zur Verfügung, damit sie sie jedes Mal verwenden können, wenn das SDK eine Anfrage vom Client ausgibt. `StaticReplayClient` AWS-Service

Note

Es gibt mehrere Möglichkeiten, den erwarteten Netzwerkverkehr zu erfassen, darunter die AWS CLI und viele Tools zur Analyse des Netzwerkverkehrs und Paket-Sniffer-Tools.

- Erstellen Sie eine Liste von `ReplayEvent` Objekten, die die erwarteten HTTP-Anfragen und die Antworten angeben, die für sie zurückgegeben werden sollen.
- Erstellen Sie eine `StaticReplayClient` mithilfe der im vorherigen Schritt erstellten HTTP-Transaktionsliste.
- Erstellen Sie ein Konfigurationsobjekt für den AWS Client und geben Sie das `StaticReplayClient` als `Config` Objekt an `http_client`.
- Erstellen Sie das AWS-Service Client-Objekt mithilfe der im vorherigen Schritt erstellten Konfiguration.
- Führen Sie die Operationen, die Sie testen möchten, mit dem Serviceobjekt aus, das für die Verwendung von konfiguriert ist `StaticReplayClient`. Jedes Mal, wenn das SDK eine API-Anfrage an sendet AWS, wird die nächste Antwort in der Liste verwendet.

Note

Die nächste Antwort in der Liste wird immer zurückgegeben, auch wenn die gesendete Anfrage nicht mit der Antwort im `ReplayEvent` Objektivektor übereinstimmt.

- Wenn alle gewünschten Anfragen gestellt wurden, rufen Sie die `StaticReplayClient.assert_requests_match()` Funktion auf, um zu überprüfen, ob die vom SDK gesendeten Anfragen mit denen in der `ReplayEvent` Objektliste übereinstimmen.

Beispiel

Schauen wir uns die Tests für dieselbe `determine_prefix_file_size()` Funktion im vorherigen Beispiel an, verwenden aber Static Replay statt Mocking.

1. Fügen Sie in einer Befehlszeile für Ihr Projektverzeichnis die [aws-smithy-runtime](#) Crate als Abhängigkeit hinzu:

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

Dadurch wird die Kiste dem `[dependencies]` Abschnitt Ihrer `Cargo.toml` Datei hinzugefügt.

2. Fügen Sie in Ihrer Quelldatei die `aws_smithy_runtime` Typen hinzu, die Sie benötigen.

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

- Der Test beginnt mit der Erstellung der `ReplayEvent` Strukturen, die jede der HTTP-Transaktionen darstellen, die während des Tests stattfinden sollen. Jedes Ereignis enthält ein HTTP-Anforderungsobjekt und ein HTTP-Antwortobjekt, die die Informationen darstellen, mit denen AWS-Service sie normalerweise antworten würden. Diese Ereignisse werden an einen Aufruf übergeben `anStaticReplayClient::new()`:

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

Das Ergebnis wird gespeichert in `replay_client`. Dies stellt einen HTTP-Client dar, der dann vom SDK für Rust verwendet werden kann, indem er in der Konfiguration des Clients angegeben wird.

- Um den Amazon S3 S3-Client zu erstellen, rufen Sie die `from_conf()` Funktion der Client-Klasse auf, um den Client mithilfe eines Konfigurationsobjekts zu erstellen:

```
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(),
);
```

Das Konfigurationsobjekt wird mit der `http_client()` Methode des Builders angegeben, und die Anmeldeinformationen werden mit der `credentials_provider()` Methode angegeben. Die Anmeldeinformationen werden mithilfe einer aufgerufenen Funktion erstellt `make_s3_test_credentials()`, die eine falsche Struktur der Anmeldeinformationen zurückgibt:

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

Diese Anmeldeinformationen müssen nicht gültig sein, da sie nicht wirklich gesendet werden AWS.

5. Führen Sie den Test aus, indem Sie die Funktion aufrufen, die getestet werden muss. In diesem Beispiel lautet der Name dieser Funktion `determine_prefix_file_size()`. Sein erster Parameter ist das Amazon S3 S3-Client-Objekt, das für seine Anfragen verwendet werden soll. Geben Sie daher den Client an, der mit dem `StaticReplayClient` so erstellt wurde, dass Anfragen von diesem bearbeitet werden, anstatt über das Netzwerk gesendet zu werden:

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

assert_eq!(19, size);

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

Wenn der Aufruf von abgeschlossen `determine_prefix_file_size()` ist, wird eine Bestätigung verwendet, um zu bestätigen, dass der zurückgegebene Wert dem erwarteten Wert entspricht. Dann wird die `StaticReplayClient` `assert_requests_match()` Methodenfunktion aufgerufen. Diese Funktion scannt die aufgezeichneten HTTP-Anfragen und bestätigt, dass sie alle mit denen übereinstimmen, die im `ReplayEvent` Objektarray angegeben wurden, das bei der Erstellung des Replay-Clients bereitgestellt wurde.

[Den vollständigen Code für diese Beispiele finden](#) Sie unter GitHub.

Waiter

Waiter sind eine clientseitige Abstraktion, die verwendet wird, um eine Ressource abzufragen, bis ein gewünschter Status erreicht ist oder bis festgestellt wird, dass die Ressource nicht in den gewünschten Zustand übergeht. Dies ist eine häufige Aufgabe, wenn Sie mit Diensten arbeiten, die irgendwann konsistent sind, wie Amazon Simple Storage Service, oder mit Diensten, die asynchron Ressourcen erstellen, wie Amazon Elastic Compute Cloud. Das Schreiben von Logik zur kontinuierlichen Abfrage des Status einer Ressource kann umständlich und fehleranfällig sein. Das Ziel der Kellner ist es, diese Verantwortung aus dem Kundencode in den Kundencode zu verlagern AWS SDK for Rust, der über fundierte Kenntnisse der zeitlichen Abläufe verfügt. AWS

AWS-Services die Kellner unterstützen, beinhalten ein `<service>::waiters` Modul.

- Das `<service>::client::Waiters` Merkmal bietet Kellnermethoden für den Kunden. Die Methoden sind für die `Client` Struktur implementiert. Alle Kellnermethoden folgen einer Standardbenennungskonvention von `wait_until_<Condition>`
 - Für Amazon S3 ist dieses Merkmal [aws_sdk_s3::client::Waiters](#).

Das folgende Beispiel verwendet Amazon S3. Die Konzepte sind jedoch für alle AWS-Service , für die ein oder mehrere Kellner definiert sind, dieselben.

Das folgende Codebeispiel zeigt die Verwendung einer Waiter-Funktion, anstatt eine Abfragelogik zu schreiben, um darauf zu warten, dass ein Bucket nach seiner Erstellung existiert.

```
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

Jede Wartemethode gibt einen Wert zurück `Result<FinalPoll<...>, WaiterError<...>>`, der verwendet werden kann, um die endgültige Antwort nach Erreichen der gewünschten Bedingung oder eines Fehlers zu ermitteln. Einzelheiten finden Sie unter [FinalPoll](#) und [WaiterError](#) in der Rust-API-Dokumentation.

SDK für Rust-Codebeispiele

Die Codebeispiele in diesem Thema zeigen Ihnen, wie Sie das AWS SDK für Rust mit verwenden AWS.

Bei Grundlagen handelt es sich um Code-Beispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Szenarios sind Code-Beispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Services aufrufen oder mit anderen AWS-Services kombinieren.

Einige Dienste enthalten zusätzliche Beispielkategorien, die zeigen, wie Sie Bibliotheken oder Funktionen nutzen können, die für den Dienst spezifisch sind.

Services

- [API Gateway Gateway-Beispiele mit SDK für Rust](#)
- [API Gateway Management API-Beispiele mit SDK für Rust](#)
- [Beispiele für Application Auto Scaling mit SDK für Rust](#)
- [Aurora-Beispiele mit SDK für Rust](#)
- [Auto Scaling Scaling-Beispiele mit SDK für Rust](#)
- [Amazon Bedrock Runtime-Beispiele mit SDK für Rust](#)
- [Beispiele für Amazon Cognito Identity Provider mit SDK für Rust](#)
- [Amazon Cognito Sync-Beispiele mit SDK für Rust](#)
- [Firehose-Beispiele mit SDK für Rust](#)
- [Amazon DocumentDB DocumentDB-Beispiele mit SDK für Rust](#)
- [DynamoDB-Beispiele mit SDK für Rust](#)
- [Amazon EBS-Beispiele mit SDK für Rust](#)
- [EC2 Amazon-Beispiele mit SDK für Rust](#)
- [Amazon ECR-Beispiele mit SDK für Rust](#)
- [Amazon ECS-Beispiele mit SDK für Rust](#)

- [Amazon EKS-Beispiele mit SDK für Rust](#)
- [AWS Glue Beispiele für die Verwendung von SDK für Rust](#)
- [IAM-Beispiele mit SDK für Rust](#)
- [AWS IoT Beispiele für die Verwendung von SDK für Rust](#)
- [Kinesis-Beispiele mit SDK für Rust](#)
- [AWS KMS Beispiele mit SDK für Rust](#)
- [Lambda-Beispiele mit SDK für Rust](#)
- [MediaLive Beispiele mit SDK für Rust](#)
- [MediaPackage Beispiele für die Verwendung von SDK für Rust](#)
- [Amazon MSK-Beispiele mit SDK für Rust](#)
- [Amazon Polly Polly-Beispiele mit SDK für Rust](#)
- [QLDB-Beispiele mit SDK für Rust](#)
- [Amazon RDS-Beispiele mit SDK für Rust](#)
- [Beispiele für Amazon RDS Data Service mit SDK für Rust](#)
- [Amazon Rekognition Rekognition-Beispiele mit SDK für Rust](#)
- [Route-53-Beispiele mit SDK für Rust](#)
- [Amazon S3 S3-Beispiele mit SDK für Rust](#)
- [SageMaker KI-Beispiele, die SDK für Rust verwenden](#)
- [Secrets Manager Manager-Beispiele mit SDK für Rust](#)
- [Amazon SES API v2-Beispiele mit SDK für Rust](#)
- [Amazon SNS SNS-Beispiele mit SDK für Rust](#)
- [Amazon SQS SQS-Beispiele mit SDK für Rust](#)
- [AWS STS Beispiele mit SDK für Rust](#)
- [Systems Manager Manager-Beispiele mit SDK für Rust](#)
- [Amazon Transcribe Transcribe-Beispiele mit SDK für Rust](#)

API Gateway Gateway-Beispiele mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe des AWS SDK für Rust mit API Gateway Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Szenarien sind Code-Beispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Services aufrufen oder mit anderen AWS-Services kombinieren.

AWS Community-Beiträge sind Beispiele, die von mehreren Teams erstellt wurden und verwaltet werden AWS. Um Feedback zu geben, verwende den Mechanismus, der in den verlinkten Repositorien zur Verfügung steht.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)
- [Szenarien](#)
- [AWS Beiträge der Community](#)

Aktionen

GetRestApis

Das folgende Codebeispiel zeigt die Verwendung `GetRestApis`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Zeigt den Amazon API Gateway Gateway-REST APIs in der Region an.

```
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(())
}
```

- Einzelheiten zur API finden Sie [GetRestApis](#) in der API-Referenz zum AWS SDK für Rust.

Szenarien

Erstellen einer Serverless-Anwendung zur Verwaltung von Fotos

Das folgende Codebeispiel zeigt, wie eine Serverless-Anwendung erstellt wird, mit der Benutzer Fotos mithilfe von Labels erstellen können.

SDK für Rust

Zeigt, wie eine Anwendung zur Verwaltung von Fotobeständen entwickelt wird, die mithilfe von Amazon Rekognition Labels in Bildern erkennt und sie für einen späteren Abruf speichert.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

Einen tiefen Einblick in den Ursprung dieses Beispiels finden Sie im Beitrag in der [AWS - Community](#).

In diesem Beispiel verwendete Dienste

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition

- Amazon S3
- Amazon SNS

AWS Beiträge der Community

Erstellen und testen Sie eine serverlose Anwendung

Das folgende Codebeispiel zeigt, wie eine serverlose Anwendung mithilfe von API Gateway mit Lambda und DynamoDB erstellt und getestet wird.

SDK für Rust

Zeigt, wie eine serverlose Anwendung, die aus einem API Gateway mit Lambda und DynamoDB besteht, mithilfe des Rust-SDK erstellt und getestet wird.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter. [GitHub](#)

In diesem Beispiel verwendete Dienste

- API Gateway
- DynamoDB
- Lambda

API Gateway Management API-Beispiele mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe des AWS SDK für Rust mit der API Gateway Management API Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

PostToConnection

Das folgende Codebeispiel zeigt die Verwendung `PostToConnection`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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);
```

- Einzelheiten zur API finden Sie [PostToConnection](#) in der API-Referenz zum AWS SDK für Rust.

Beispiele für Application Auto Scaling mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe des AWS SDK für Rust mit Application Auto Scaling Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

DescribeScalingPolicies

Das folgende Codebeispiel zeigt die Verwendung `DescribeScalingPolicies`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

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

Ok(())
}
```

- Einzelheiten zur API finden Sie [DescribeScalingPolicies](#) in der API-Referenz zum AWS SDK für Rust.

Aurora-Beispiele mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe des AWS SDK für Rust mit Aurora Aktionen ausführen und allgemeine Szenarien implementieren.

Bei Grundlagen handelt es sich um Code-Beispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hello Aurora

Die folgenden Codebeispiele veranschaulichen die ersten Schritte mit Aurora.

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
use aws_sdk_rds::Client;
```

```

#[derive(Debug)]
struct Error(String);
impl std::fmt::Display for Error {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(f, "{}", self.0)
    }
}
impl std::error::Error for Error {}

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

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

    Ok(())
}

```

- Einzelheiten zur API finden Sie unter [DBClustersDescribe](#) in AWS SDK for Rust API-Referenz.

Themen

- [Grundlagen](#)
- [Aktionen](#)

Grundlagen

Erlernen der Grundlagen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie eine benutzerdefinierte Aurora-DB-Cluster-Parametergruppe und legen Sie Parameterwerte fest.
- Erstellen Sie einen DB-Cluster, der die Parametergruppe verwendet.
- Erstellen Sie eine DB-Instance, die eine Datenbank enthält.
- Erstellen Sie einen Snapshot des DB-Clusters und bereinigen Sie dann die Ressourcen.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Eine Bibliothek, die die szenariospezifischen Funktionen für das Aurora-Szenario enthält.

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

use aws_sdk_rds::{
    error::ProvideErrorMetadata,

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

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

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

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

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

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

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



```

}

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

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

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

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

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

```

```

}

impl From<aws_sdk_rds::types::Parameter> for AuroraScenarioParameter {
    fn from(value: aws_sdk_rds::types::Parameter) -> Self {
        AuroraScenarioParameter {
            name: value.parameter_name.unwrap_or_default(),
            allowed_values: value.allowed_values.unwrap_or_default(),
            current_value: value.parameter_value.unwrap_or_default(),
        }
    }
}

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

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

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

```

```

    let describe_db_engine_versions =
self.rds.describe_db_engine_versions(DB_ENGINE).await;
    trace!(versions=?describe_db_engine_versions, "full list of versions");

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

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

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

    Ok(versions)
}

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

```

```

        .as_str(),
    )
    .await;

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

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

    match create_db_cluster_parameter_group {
        Ok(CreateDbClusterParameterGroupOutput {
            db_cluster_parameter_group: None,
            ..
        }) => {
            return Err(ScenarioError::with(
                "CreateDBClusterParameterGroup had empty response",
            ));
        }
        Err(error) => {
            if error.code() == Some("DBParameterGroupAlreadyExists") {
                info!("Cluster Parameter Group already exists, nothing to do");
            } else {

```

```

        return Err(ScenarioError::new(
            "Could not create Cluster Parameter Group",
            &error,
        ));
    }
}
_ => {
    info!("Created Cluster Parameter Group");
}
}

Ok(())
}

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

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

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

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

```

```

    }

    let db_cluster = describe_db_clusters_output
        .unwrap()
        .db_clusters
        .and_then(|output| output.first().cloned());

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

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

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

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

    Ok(parameters)
}

```

```

// Modify both the auto_increment_offset and auto_increment_increment parameters
in one call in the custom parameter group. Set their ParameterValue fields to a new
allowable value. rds.ModifyDbClusterParameterGroup.
pub async fn update_auto_increment(
    &self,
    offset: u8,
    increment: u8,
) -> Result<(), ScenarioError> {
    let modify_db_cluster_parameter_group = self
        .rds
        .modify_db_cluster_parameter_group(
            DB_CLUSTER_PARAMETER_GROUP_NAME,
            vec![
                Parameter::builder()
                    .parameter_name("auto_increment_offset")
                    .parameter_value(format!("{offset}"))
                    .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                    .build(),
                Parameter::builder()
                    .parameter_name("auto_increment_increment")
                    .parameter_value(format!("{increment}"))
                    .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                    .build(),
            ],
        )
        .await;

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

    Ok(())
}

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

```

```
// Get a list of instance classes available for the selected engine and engine
version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql', EngineVersion=).

// Create a database instance in the cluster.
// Wait for DB instance to be ready. Call rds.DescribeDbInstances and check for
DBInstanceStatus == 'available'.
pub async fn start_cluster_and_instance(&mut self) -> Result<(), ScenarioError>
{
    if self.password.is_none() {
        return Err(ScenarioError::with(
            "Must set Secret Password before starting a cluster",
        ));
    }
    let create_db_cluster = self
        .rds
        .create_db_cluster(
            DB_CLUSTER_IDENTIFIER,
            DB_CLUSTER_PARAMETER_GROUP_NAME,
            DB_ENGINE,
            self.engine_version.as_deref().expect("engine version"),
            self.username.as_deref().expect("username"),
            self.password
                .replace(SecretString::new("").to_string())
                .expect("password"),
        )
        .await;
    if let Err(err) = create_db_cluster {
        return Err(ScenarioError::new(
            "Failed to create DB Cluster with cluster group",
            &err,
        ));
    }

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

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

    info!(
```



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

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

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

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

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

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

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

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

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

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

```

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

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

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

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

    // Delete the instance. rds.DeleteDbInstance.
    let delete_db_instance = self
        .rds
        .delete_db_instance(
            self.db_instance_identifier
                .as_deref()
                .expect("instance identifier"),
        )
        .await;
    if let Err(err) = delete_db_instance {
        let identifier = self
            .db_instance_identifier
            .as_deref()
            .unwrap_or("Missing Instance Identifier");

```

```

        let message = format!("failed to delete db instance {identifier}");
        clean_up_errors.push(ScenarioError::new(message, &err));
    } else {
        // Wait for the instance to delete
        let waiter = Waiter::default();
        while waiter.sleep().await.is_ok() {
            let describe_db_instances = self.rds.describe_db_instances().await;
            if let Err(err) = describe_db_instances {
                clean_up_errors.push(ScenarioError::new(
                    "Failed to check instance state during deletion",
                    &err,
                ));
                break;
            }
            let db_instances = describe_db_instances
                .unwrap()
                .db_instances()
                .iter()
                .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
                .cloned()
                .collect:::<Vec<DbInstance>>();

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

    // Delete the DB cluster. rds.DeleteDbCluster.
    let delete_db_cluster = self
        .rds

```

```

        .delete_db_cluster(
            self.db_cluster_identifier
                .as_deref()
                .expect("cluster identifier"),
        )
        .await;

    if let Err(err) = delete_db_cluster {
        let identifier = self
            .db_cluster_identifier
            .as_deref()
            .unwrap_or("Missing DB Cluster Identifier");
        let message = format!("failed to delete db cluster {identifier}");
        clean_up_errors.push(ScenarioError::new(message, &err));
    } else {
        // Wait for the instance and cluster to fully delete.
        rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
        let waiter = Waiter::default();
        while waiter.sleep().await.is_ok() {
            let describe_db_clusters = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;
            if let Err(err) = describe_db_clusters {
                clean_up_errors.push(ScenarioError::new(
                    "Failed to check cluster state during deletion",
                    &err,
                ));
                break;
            }
            let describe_db_clusters = describe_db_clusters.unwrap();
            let db_clusters = describe_db_clusters.db_clusters();
            if db_clusters.is_empty() {
                trace!("Delete cluster waited and no clusters were found");
                break;
            }
            match db_clusters.first().unwrap().status() {
                Some("Deleting") => continue,
                Some(status) => {
                    info!("Attempting to delete but clusters is in {status}");
                }
            }
        }
    }

```

```
        continue;
    }
    None => {
        warn!("No status for DB cluster");
        break;
    }
}
}
}

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

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

#[cfg(test)]
pub mod tests;
```

Tests für die Bibliothek, die Automocks für den RDS-Client-Wrapper nutzen.

```
use crate::rds::MockRdsImpl;

use super::*;

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

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

describe_orderable_db_instance_options::DescribeOrderableDBInstanceOptionsError,
        modify_db_cluster_parameter_group::{
            ModifyDBClusterParameterGroupError, ModifyDbClusterParameterGroupOutput,
        },
    },
    types::{
        error::DbParameterGroupAlreadyExistsFault, DbClusterEndpoint,
        DbEngineVersion,
        OrderableDbInstanceOption,
    },
};
use aws_smithy_runtime_api::http::{Response, StatusCode};
```

```
use aws_smithy_types::body::SdkBody;
use mockall::predicate::eq;
use secrecy::ExposeSecret;

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

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

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

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

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

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

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

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



```

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

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

    assert!(set_engine.is_err());
}

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

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

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

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

    assert!(set_engine.is_err());
}

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

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

```

```

        .engine_version("f1a")
        .build(),
    )
    .db_engine_versions(
        DbEngineVersion::builder()
            .db_parameter_group_family("f1")
            .engine_version("f1b")
            .build(),
    )
    .db_engine_versions(
        DbEngineVersion::builder()
            .db_parameter_group_family("f2")
            .engine_version("f2a")
            .build(),
    )
    .db_engine_versions(DbEngineVersion::builder().build())
    .build()
});

let scenario = AuroraScenario::new(mock_rds);

let versions_map = scenario.get_engines().await;

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

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

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

```

```

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

let scenario = AuroraScenario::new(mock_rds);

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

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

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

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

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

```

```

        .build(),
        OrderableDbInstanceOption::builder()
            .db_instance_class("t3")
            .storage_type("aurora")
            .build(),
    ])
});

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

let instance_classes = scenario.get_instance_classes().await;

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

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

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

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

```

```
let instance_classes = scenario.get_instance_classes().await;

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

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

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

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

    assert!(cluster.is_ok());
}

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

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

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

    mock_rds
        .expect_describe_db_clusters()
```

```

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

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

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

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

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

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

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

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

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

```

```

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

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

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

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

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

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

```

```

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

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

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

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

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

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

```



```

    assert_matches!(params, Err(ScenarioError { message, context: _ }) if message ==
    "Failed to retrieve parameters for RustSDKCodeExamplesDBParameterGroup");
}

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

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

    let scenario = AuroraScenario::new(mock_rds);

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

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

```

```

mock_rds
    .expect_modify_db_cluster_parameter_group()
    .return_once(|_, _| {
        Err(SdkError::service_error(
            ModifyDBClusterParameterGroupError::unhandled(Box::new(Error::new(
                ErrorKind::Other,
                "modify_db_cluster_parameter_group_error",
            ))),
            Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
        ))
    });

let scenario = AuroraScenario::new(mock_rds);

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

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

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

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");

```

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

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

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

mock_rds
    .expect_describe_db_cluster_endpoints()
    .with(eq("RustSDKCodeExamplesDBCluster"))
    .return_once(|_| {
```

```

        Ok(DescribeDbClusterEndpointsOutput::builder()

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

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

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

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

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

```

```

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

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

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

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

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

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

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

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

    mock_rds
        .expect_create_db_cluster()

```

```

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

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

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

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

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

    mock_rds
        .expect_create_db_cluster()

```

```

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

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

mock_rds
    .expect_describe_db_clusters()
    .with(eq("RustSDKCodeExamplesDBCluster"))
    .times(1)
    .returning(|_| {
        Err(SdkError::service_error(
            DescribeDBClustersError::unhandled(Box::new(Error::new(
                ErrorKind::Other,
            )))
        ))
    });

```

```

        "describe cluster error",
    )))
    Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
    ))
})
.with(eq("RustSDKCodeExamplesDBCluster"))
.times(1)
.returning(|id| {
    Ok(DescribeDbClustersOutput::builder()
        .db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
        .build())
});

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

mock_rds
    .expect_describe_db_cluster_endpoints()
    .return_once(|_| {
        Ok(DescribeDbClusterEndpointsOutput::builder()
            .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
            .build())
        });

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

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

```



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

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

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

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

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

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
```

```

        Ok(DescribeDbClustersOutput::builder()
            .db_clusters(
                DbCluster::builder()
                    .db_cluster_idenfifier(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_idenfifier = Some(String::from("MockCluster"));
scenario.db_instance_idenfifier = Some(String::from("MockInstance"));
scenario.db_cluster_parameter_group = Some(
    DbClusterParameterGroup::builder()
        .db_cluster_parameter_group_name("MockParamGroup")
        .build(),
);

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

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

```

```
}

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

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

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

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

    mock_rds
        .expect_describe_db_clusters()

```

```

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

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

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

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

```

```

        assert_eq!(errs.len(), 2);
        assert_matches!(errs.first(), Some(ScenarioError {message, context: _}) if
message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
message == "Failed to check cluster state during deletion");
    });

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

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

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

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

```

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

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

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

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

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

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

Eine Binärdatei zum vollständigen Ausführen des Szenarios, wobei Inquirer verwendet wird, damit der Benutzer einige Entscheidungen treffen kann.

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

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

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

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

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

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

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

```

}

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

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

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

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

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

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

```



```

    }

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

    Ok(scenario)
}

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

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

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

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

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

```

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

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

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

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

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

    Ok(())
}
```

```

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

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

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

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

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

    Ok(())
}

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

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

    if prepare_cluster(&mut scenario, &mut warnings).await.is_ok() {
        println!("Configured database cluster, starting an instance.");
    }
}

```

```
        if let Err(err) = run_instance(&mut scenario).await {
            warnings.push("Problem running instance", err);
        }
    }

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

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

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

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

    match parameters {
        Ok(parameters) => {
            println!("Current parameters");
        }
    }
}
```

```

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

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

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

```

Ein Wrapper für den Service Amazon RDS, der Automocking für Tests ermöglicht.

```

use aws_sdk_rds::{
    error::SdkError,

```

```

operation::{
    create_db_cluster::{CreateDBClusterError, CreateDbClusterOutput},
    create_db_cluster_parameter_group::{CreateDBClusterParameterGroupError,
    create_db_cluster_parameter_group::{CreateDbClusterParameterGroupOutput,
CreateDbClusterSnapshotOutput},
    create_db_instance::{CreateDBInstanceError, CreateDbInstanceOutput},
    delete_db_cluster::{DeleteDBClusterError, DeleteDbClusterOutput},
    delete_db_cluster_parameter_group::{
        DeleteDBClusterParameterGroupError, DeleteDbClusterParameterGroupOutput,
    },
    delete_db_instance::{DeleteDBInstanceError, DeleteDbInstanceOutput},
    describe_db_cluster_endpoints::{
        DescribeDBClusterEndpointsError, DescribeDbClusterEndpointsOutput,
    },
    describe_db_cluster_parameters::{
        DescribeDBClusterParametersError, DescribeDbClusterParametersOutput,
    },
    describe_db_clusters::{DescribeDBClustersError, DescribeDbClustersOutput},
    describe_db_engine_versions::{
        DescribeDBEngineVersionsError, DescribeDbEngineVersionsOutput,
    },
    describe_db_instances::{DescribeDBInstancesError,
DescribeDbInstancesOutput},

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

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

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

pub struct RdsImpl {

```

```
    pub inner: RdsClient,
}

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

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

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

    pub async fn create_db_cluster_parameter_group(
        &self,
        name: &str,
        description: &str,
        family: &str,
```

```
    ) -> Result<CreateDbClusterParameterGroupOutput,  
SdkError<CreateDBClusterParameterGroupError>>  
    {  
        self.inner  
            .create_db_cluster_parameter_group()  
            .db_cluster_parameter_group_name(name)  
            .description(description)  
            .db_parameter_group_family(family)  
            .send()  
            .await  
    }  
  
pub async fn describe_db_clusters(  
    &self,  
    id: &str,  
    ) -> Result<DescribeDbClustersOutput, SdkError<DescribeDBClustersError>> {  
    self.inner  
        .describe_db_clusters()  
        .db_cluster_identifier(id)  
        .send()  
        .await  
    }  
  
pub async fn describe_db_cluster_parameters(  
    &self,  
    name: &str,  
    ) -> Result<Vec<DescribeDbClusterParametersOutput>,  
SdkError<DescribeDBClusterParametersError>>  
    {  
        self.inner  
            .describe_db_cluster_parameters()  
            .db_cluster_parameter_group_name(name)  
            .into_paginator()  
            .send()  
            .try_collect()  
            .await  
    }  
  
pub async fn modify_db_cluster_parameter_group(  
    &self,  
    name: &str,  
    parameters: Vec<Parameter>,  
    ) -> Result<ModifyDbClusterParameterGroupOutput,  
SdkError<ModifyDBClusterParameterGroupError>>
```



```
{
    self.inner
        .modify_db_cluster_parameter_group()
        .db_cluster_parameter_group_name(name)
        .set_parameters(Some(parameters))
        .send()
        .await
}

pub async fn create_db_cluster(
    &self,
    name: &str,
    parameter_group: &str,
    engine: &str,
    version: &str,
    username: &str,
    password: SecretString,
) -> Result<CreateDbClusterOutput, SdkError<CreateDBClusterError>> {
    self.inner
        .create_db_cluster()
        .db_cluster_idenfier(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_idenfier(cluster_name)
        .db_instance_idenfier(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_identifiler: &str,
) -> Result<DeleteDbInstanceOutput, SdkError<DeleteDBInstanceError>> {
    self.inner
        .delete_db_instance()
        .db_instance_identifiler(instance_identifiler)
        .skip_final_snapshot(true)
        .send()
        .await
}

pub async fn delete_db_cluster(
    &self,
    cluster_identifiler: &str,
) -> Result<DeleteDbClusterOutput, SdkError<DeleteDBClusterError>> {
    self.inner
        .delete_db_cluster()
        .db_cluster_identifiler(cluster_identifiler)
        .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
}
}
```

Die Datei „Cargo.toml“ mit in diesem Szenario verwendeten Abhängigkeiten.

```
[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"] }
```

- Weitere API-Informationen finden Sie in den folgenden Themen der API-Referenz zu AWS - SDK für Rust.
 - [CreateDBCluster](#)
 - [CreateDBClusterParameterGroup](#)
 - [DBClusterSchnappschuss erstellen](#)
 - [CreateDBInstance](#)
 - [LöschenDBCluster](#)
 - [LöschenDBClusterParameterGroup](#)
 - [LöschenDBInstance](#)
 - [Beschreiben DBCluster ParameterGroups](#)
 - [Beschreiben Sie die DBCluster Parameter](#)

- [Beschreiben Sie DBCluster Schnappschüsse](#)
- [Beschreiben DBClusters](#)
- [DBEngineVersionen beschreiben](#)
- [Beschreiben DBInstances](#)
- [DescribeOrderableDBInstanceOptionen](#)
- [Modifizieren SieDBClusterParameterGroup](#)

Aktionen

CreateDBCluster

Das folgende Codebeispiel zeigt die Verwendung `CreateDBCluster`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// 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_idenfifier = create_db_instance
    .unwrap()
    .db_instance
    .and_then(|i| i.db_instance_idenfifier);

// 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_idenfifier
                .as_deref()
                .expect("cluster idenfifier"),
        )
        .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_idenfifier
                .as_deref()
                .expect("instance idenfifier"),
        )
        .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_identifizier(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_identifizier(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;
}

```

- Einzelheiten zur API finden Sie in der Referenz „[Create DBCluster](#) in AWS SDK for Rust API“.

CreateDBClusterParameterGroup

Das folgende Codebeispiel zeigt, wie man es benutzt `CreateDBClusterParameterGroup`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

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

    match create_db_cluster_parameter_group {
        Ok(CreateDbClusterParameterGroupOutput {
            db_cluster_parameter_group: None,
            ..
        }) => {
            return Err(ScenarioError::with(

```



```

        "CreateDBClusterParameterGroup had empty response",
    ));
    });
}
Err(error) => {
    if error.code() == Some("DBParameterGroupAlreadyExists") {
        info!("Cluster Parameter Group already exists, nothing to do");
    } else {
        return Err(ScenarioError::new(
            "Could not create Cluster Parameter Group",
            &error,
        ));
    }
}
_ => {
    info!("Created Cluster Parameter Group");
}
}

Ok(())
}

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

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

    mock_rds
        .expect_create_db_cluster_parameter_group()

```

```

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

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

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

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

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

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

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

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

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

    assert!(set_engine.is_err());
}

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

```

```
let mut mock_rds = MockRdsImpl::default();

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

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

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

assert!(set_engine.is_err());
}
```

- Einzelheiten zur API finden Sie in der Referenz „[Create DBCluster ParameterGroup](#) in AWS SDK for Rust API“.

CreateDBClusterSnapshot

Das folgende Codebeispiel zeigt, wie man es benutzt CreateDBClusterSnapshot.

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
```

```
// Create an Aurora DB cluster database cluster that contains a MySQL database
and uses the parameter group you created.
// Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
// Get a list of instance classes available for the selected engine and engine
version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql', EngineVersion=).

// Create a database instance in the cluster.
// Wait for DB instance to be ready. Call rds.DescribeDbInstances and check for
DBInstanceStatus == 'available'.
pub async fn start_cluster_and_instance(&mut self) -> Result<(), ScenarioError>
{
    if self.password.is_none() {
        return Err(ScenarioError::with(
            "Must set Secret Password before starting a cluster",
        ));
    }
    let create_db_cluster = self
        .rds
        .create_db_cluster(
            DB_CLUSTER_IDENTIFIER,
            DB_CLUSTER_PARAMETER_GROUP_NAME,
            DB_ENGINE,
            self.engine_version.as_deref().expect("engine version"),
            self.username.as_deref().expect("username"),
            self.password
                .replace(SecretString::new("").to_string())
                .expect("password"),
        )
        .await;
    if let Err(err) = create_db_cluster {
        return Err(ScenarioError::new(
            "Failed to create DB Cluster with cluster group",
            &err,
        ));
    }

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

    if self.db_cluster_identifier.is_none() {
```

```

        return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
    }

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

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

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

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

```

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

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

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

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

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

let endpoints_available = endpoints
```

```
        .unwrap()
        .db_cluster_endpoints()
        .iter()
        .all(|endpoint| endpoint.status() == Some("available"));

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

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

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

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

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

```
        .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
        .build()
    });

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

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

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



```

        )
        .build()
    });

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

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

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

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

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

```

```

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

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

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

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

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

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

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

```

```

}

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

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

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

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

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

```

```
}

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

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

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

```

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

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

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

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

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

```

```

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

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

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

```

- Einzelheiten zur API finden Sie in der API-Referenz „DBClusterSnapshot im AWS SDK für Rust [erstellen](#)“.

CreateDBInstance

Das folgende Codebeispiel zeigt die Verwendung `CreateDBInstance`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

// 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;
}

```

- Einzelheiten zur API finden Sie in der Referenz „[Create DBInstance](#) in AWS SDK for Rust API“.

DeleteDBCluster

Das folgende Codebeispiel zeigt, wie man es benutzt DeleteDBCluster.

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

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_identifrier
                .as_deref()
                .expect("instance identifrier"),
        )
        .await;
    if let Err(err) = delete_db_instance {
        let identifrier = self
            .db_instance_identifrier
            .as_deref()
            .unwrap_or("Missing Instance Identifrier");
    }
}

```

```
        let message = format!("failed to delete db instance {identifier}");
        clean_up_errors.push(ScenarioError::new(message, &err));
    } else {
        // Wait for the instance to delete
        let waiter = Waiter::default();
        while waiter.sleep().await.is_ok() {
            let describe_db_instances = self.rds.describe_db_instances().await;
            if let Err(err) = describe_db_instances {
                clean_up_errors.push(ScenarioError::new(
                    "Failed to check instance state during deletion",
                    &err,
                ));
                break;
            }
            let db_instances = describe_db_instances
                .unwrap()
                .db_instances()
                .iter()
                .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
                .cloned()
                .collect::<Vec<DbInstance>>();

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

    // Delete the DB cluster. rds.DeleteDbCluster.
    let delete_db_cluster = self
        .rds
```

```

        .delete_db_cluster(
            self.db_cluster_identifier
                .as_deref()
                .expect("cluster identifier"),
        )
        .await;

    if let Err(err) = delete_db_cluster {
        let identifier = self
            .db_cluster_identifier
            .as_deref()
            .unwrap_or("Missing DB Cluster Identifier");
        let message = format!("failed to delete db cluster {identifier}");
        clean_up_errors.push(ScenarioError::new(message, &err));
    } else {
        // Wait for the instance and cluster to fully delete.
        rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
        let waiter = Waiter::default();
        while waiter.sleep().await.is_ok() {
            let describe_db_clusters = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;
            if let Err(err) = describe_db_clusters {
                clean_up_errors.push(ScenarioError::new(
                    "Failed to check cluster state during deletion",
                    &err,
                ));
                break;
            }
            let describe_db_clusters = describe_db_clusters.unwrap();
            let db_clusters = describe_db_clusters.db_clusters();
            if db_clusters.is_empty() {
                trace!("Delete cluster waited and no clusters were found");
                break;
            }
            match db_clusters.first().unwrap().status() {
                Some("Deleting") => continue,
                Some(status) => {
                    info!("Attempting to delete but clusters is in {status}");
                }
            }
        }
    }

```

```

        continue;
    }
    None => {
        warn!("No status for DB cluster");
        break;
    }
}
}
}

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

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

pub async fn delete_db_cluster(
    &self,
    cluster_identifiler: &str,
) -> Result<DeleteDbClusterOutput, SdkError<DeleteDBClusterError>> {
    self.inner
        .delete_db_cluster()

```

```
        .db_cluster_identififier(cluster_identififier)
        .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_identififier("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_identififier(id)
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbClustersOutput::builder().build()));
}
```

```

        .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;
}

```

- Einzelheiten zur API finden Sie unter API-Referenz zum [Löschen DBCluster](#) im AWS SDK für Rust.

DeleteDBClusterParameterGroup

Das folgende Codebeispiel zeigt die Verwendung `DeleteDBClusterParameterGroup`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

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

    // Delete the instance. rds.DeleteDbInstance.
    let delete_db_instance = self
        .rds
        .delete_db_instance(

```

```

        self.db_instance_identifier
            .as_deref()
            .expect("instance identifier"),
    )
    .await;
if let Err(err) = delete_db_instance {
    let identifier = self
        .db_instance_identifier
        .as_deref()
        .unwrap_or("Missing Instance Identifier");
    let message = format!("failed to delete db instance {identifier}");
    clean_up_errors.push(ScenarioError::new(message, &err));
} else {
    // Wait for the instance to delete
    let waiter = Waiter::default();
    while waiter.sleep().await.is_ok() {
        let describe_db_instances = self.rds.describe_db_instances().await;
        if let Err(err) = describe_db_instances {
            clean_up_errors.push(ScenarioError::new(
                "Failed to check instance state during deletion",
                &err,
            ));
            break;
        }
        let db_instances = describe_db_instances
            .unwrap()
            .db_instances()
            .iter()
            .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
            .cloned()
            .collect:::<Vec<DbInstance>>();

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

```

```

        warn!("No status for DB instance");
        break;
    }
}
}

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

if let Err(err) = delete_db_cluster {
    let identifier = self
        .db_cluster_identifier
        .as_deref()
        .unwrap_or("Missing DB Cluster Identifier");
    let message = format!("failed to delete db cluster {identifier}");
    clean_up_errors.push(ScenarioError::new(message, &err));
} else {
    // Wait for the instance and cluster to fully delete.
    rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
    let waiter = Waiter::default();
    while waiter.sleep().await.is_ok() {
        let describe_db_clusters = self
            .rds
            .describe_db_clusters(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;
        if let Err(err) = describe_db_clusters {
            clean_up_errors.push(ScenarioError::new(
                "Failed to check cluster state during deletion",
                &err,
            ));
            break;
        }
    }
}

```

```

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

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

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

```

```

        Err(clean_up_errors)
    }
}

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

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

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

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

```

```

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

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

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

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

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

```

```

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

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

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

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

```

```

    ))
  });

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

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

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

let mut scenario = AuroraScenario::new(mock_rds);
scenario.db_cluster_identifier = Some(String::from("MockCluster"));
scenario.db_instance_identifier = Some(String::from("MockInstance"));
scenario.db_cluster_parameter_group = Some(

```



```

        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

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

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

```

- Einzelheiten zur API finden Sie unter API-Referenz zum [Löschen DBCluster ParameterGroup](#) im AWS SDK für Rust.

DeleteDBInstance

Das folgende Codebeispiel zeigt die Verwendung `DeleteDBInstance`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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_identifer
                .as_deref()
                .expect("instance identifier"),
        )
        .await;
    if let Err(err) = delete_db_instance {
        let identifier = self
            .db_instance_identifer
            .as_deref()
            .unwrap_or("Missing Instance Identifier");
        let message = format!("failed to delete db instance {identifier}");
        clean_up_errors.push(ScenarioError::new(message, &err));
    } else {
        // Wait for the instance to delete
        let waiter = Waiter::default();
        while waiter.sleep().await.is_ok() {
            let describe_db_instances = self.rds.describe_db_instances().await;
            if let Err(err) = describe_db_instances {
                clean_up_errors.push(ScenarioError::new(
                    "Failed to check instance state during deletion",
                    &err,
                ));
                break;
            }
        }
        let db_instances = describe_db_instances
            .unwrap()
            .db_instances()
```

```

        .iter()
        .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
        .cloned()
        .collect::<Vec<DbInstance>>());

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

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

if let Err(err) = delete_db_cluster {
    let identifier = self
        .db_cluster_identifier
        .as_deref()
        .unwrap_or("Missing DB Cluster Identifier");
    let message = format!("failed to delete db cluster {identifier}");
    clean_up_errors.push(ScenarioError::new(message, &err));
} else {
    // Wait for the instance and cluster to fully delete.
    rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.

```

```

    let waiter = Waiter::default();
    while waiter.sleep().await.is_ok() {
        let describe_db_clusters = self
            .rds
            .describe_db_clusters(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;
        if let Err(err) = describe_db_clusters {
            clean_up_errors.push(ScenarioError::new(
                "Failed to check cluster state during deletion",
                &err,
            ));
            break;
        }
        let describe_db_clusters = describe_db_clusters.unwrap();
        let db_clusters = describe_db_clusters.db_clusters();
        if db_clusters.is_empty() {
            trace!("Delete cluster waited and no clusters were found");
            break;
        }
        match db_clusters.first().unwrap().status() {
            Some("Deleting") => continue,
            Some(status) => {
                info!("Attempting to delete but clusters is in {status}");
                continue;
            }
            None => {
                warn!("No status for DB cluster");
                break;
            }
        }
    }
}

// Delete the DB cluster parameter group. rds.DeleteDbClusterParameterGroup.
let delete_db_cluster_parameter_group = self
    .rds
    .delete_db_cluster_parameter_group(
        self.db_cluster_parameter_group
            .map(|g| {
                g.db_cluster_parameter_group_name
            })
    )

```

```

                .unwrap_or_else(||
DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                })
                .as_deref()
                .expect("cluster parameter group name"),
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
                &error,
            ))
        }

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

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

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

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

    mock_rds
        .expect_describe_db_instances()

```

```

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

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

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

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

```

```

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

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

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

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

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

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(

```

```

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

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

mock_rds
    .expect_describe_db_clusters()
    .with(eq("MockCluster"))
    .times(1)
    .returning(|id| {
        Ok(DescribeDbClustersOutput::builder()
            .db_clusters(
                DbCluster::builder()
                    .db_cluster_identifier(id)
                    .status("Deleting")
                    .build(),
            )
            .build())
    })
    .with(eq("MockCluster"))
    .times(1)
    .returning(|_| {
        Err(SdkError::service_error(
            DescribeDBClustersError::unhandled(Box::new(Error::new(
                ErrorKind::Other,

```



```

        "describe db clusters error",
    )))
    Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
    ))
});

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

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

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

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

```

- Einzelheiten zur API finden Sie unter API-Referenz zum [Löschen DBInstance](#) im AWS SDK für Rust.

DescribeDBClusterParameters

Das folgende Codebeispiel zeigt die Verwendung `DescribeDBClusterParameters`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

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

    let parameters = parameters_output
        .unwrap()
        .into_iter()
```

```

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

    Ok(parameters)
}

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

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

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

```

```

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

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

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

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

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

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

```

- Einzelheiten zur API finden Sie unter [DBClusterDescribe Parameters](#) in AWS SDK for Rust API-Referenz.

DescribeDBClusters

Das folgende Codebeispiel zeigt die Verwendung `DescribeDBClusters`.

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// 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_idenfier = create_db_cluster
    .unwrap()
    .db_cluster
    .and_then(|c| c.db_cluster_idenfier);

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

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

let create_db_instance = self
    .rds
    .create_db_instance(
        self.db_cluster_idenfier.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_identifizier = create_db_instance
    .unwrap()
    .db_instance
    .and_then(|i| i.db_instance_identifizier);

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

let instances_available = instance
    .unwrap()
    .db_instances()
    .iter()
```

```
        .all(|instance| instance.db_instance_status() == Some("Available"));

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

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

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

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

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

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

#[tokio::test]
```



```
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

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

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

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
```

```

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

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

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

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

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

tokio::time::pause();
let assertions = tokio::spawn(async move {
    let create = scenario.start_cluster_and_instance().await;
    assert!(create.is_ok());
    assert!(scenario
        .password
        .replace(SecretString::new("BAD SECRET".into()))
        .unwrap()

```

```

        .expose_secret()
        .is_empty());
    assert_eq!(
        scenario.db_cluster_identifier,
        Some("RustSDKCodeExamplesDBCluster".into())
    );
});
tokio::time::advance(Duration::from_secs(1)).await;
tokio::time::resume();
let _ = assertions.await;
}

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

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

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

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

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

    mock_rds

```

```

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

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

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

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

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

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(

```

```

        ErrorKind::Other,
        "create db instance error",
    )),
    Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
))
});

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

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

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

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

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
        });
}

```

```

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

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

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

```

```

        )
        .build()
    });

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

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

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

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

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

```

- Einzelheiten zur API finden Sie unter [DBClustersDescribe](#) in AWS SDK for Rust API-Referenz.

DescribeDBEngineVersions

Das folgende Codebeispiel zeigt, wie man es benutzt DescribeDBEngineVersions.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// 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"
    );
}

```

```
}

```

- Einzelheiten zur API finden Sie unter [DBEngineVersionen im AWS SDK für die Rust-API-Referenz beschreiben](#).

DescribeDBInstances

Das folgende Codebeispiel zeigt die Verwendung `DescribeDBInstances`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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_identifiser ==
self.db_cluster_identifiser)
        .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_identifiser
            .as_deref()
            .expect("cluster identifiser"),
    )
    .await;
```

```

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

```

```

        }
    }
}

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

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

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

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

    mock_rds
        .expect_delete_db_instance()

```

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

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

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

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

```

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

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

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

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

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

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

    mock_rds
        .expect_describe_db_instances()

```



```

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

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

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

```

```

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

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

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

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

tokio::time::advance(Duration::from_secs(1)).await; // Wait for first Describe
Instances
tokio::time::advance(Duration::from_secs(1)).await; // Wait for second Describe
Instances
tokio::time::advance(Duration::from_secs(1)).await; // Wait for first Describe
Cluster

```

```

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

```

- Einzelheiten zur API finden Sie unter [DBInstancesDescribe](#) in AWS SDK for Rust API-Referenz.

DescribeOrderableDBInstanceOptions

Das folgende Codebeispiel zeigt, wie man es benutzt `DescribeOrderableDBInstanceOptions`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

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

    describe_orderable_db_instance_options_items
        .map(|options| {
            options
                .iter()
                .filter(|o| o.storage_type() == Some("aurora"))
                .map(|o| o.db_instance_class().unwrap_or_default().to_string())
                .collect:::<Vec<String>>()
        })
}

```

```

        .map_err(|err| ScenarioError::new("Could not get available instance
classes", &err))
    }

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

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

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

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

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

```

```

        OrderableDbInstanceOption::builder()
            .db_instance_class("t1")
            .storage_type("aurora-iopt1")
            .build(),
        OrderableDbInstanceOption::builder()
            .db_instance_class("t2")
            .storage_type("aurora")
            .build(),
        OrderableDbInstanceOption::builder()
            .db_instance_class("t3")
            .storage_type("aurora")
            .build(),
    ])
});

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

let instance_classes = scenario.get_instance_classes().await;

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

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

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

```

```

        ))
    });

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

    let instance_classes = scenario.get_instance_classes().await;

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

```

- Einzelheiten zur API finden Sie in der Referenz zu den [DescribeOrderableDBInstanceOptionen](#) im AWS SDK für die Rust-API.

ModifyDBClusterParameterGroup

Das folgende Codebeispiel zeigt, wie man es benutzt `ModifyDBClusterParameterGroup`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

// Modify both the auto_increment_offset and auto_increment_increment parameters
in one call in the custom parameter group. Set their ParameterValue fields to a new
allowable value. rds.ModifyDbClusterParameterGroup.
pub async fn update_auto_increment(
    &self,
    offset: u8,
    increment: u8,
) -> Result<(), ScenarioError> {
    let modify_db_cluster_parameter_group = self

```

```

        .rds
        .modify_db_cluster_parameter_group(
            DB_CLUSTER_PARAMETER_GROUP_NAME,
            vec![
                Parameter::builder()
                    .parameter_name("auto_increment_offset")
                    .parameter_value(format!("{offset}"))
                    .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                    .build(),
                Parameter::builder()
                    .parameter_name("auto_increment_increment")
                    .parameter_value(format!("{increment}"))
                    .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                    .build(),
            ],
        )
        .await;

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

    Ok(())
}

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

#[tokio::test]

```

```

async fn test_scenario_update_auto_increment() {
    let mut mock_rds = MockRdsImpl::default();

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

    let scenario = AuroraScenario::new(mock_rds);

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

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

    mock_rds
        .expect_modify_db_cluster_parameter_group()
        .return_once(|_, _| {
            Err(SdkError::service_error(
                ModifyDBClusterParameterGroupError::unhandled(Box::new(Error::new(

```



```
                ErrorKind::Other,
                "modify_db_cluster_parameter_group_error",
            )))
            Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
        ));

    let scenario = AuroraScenario::new(mock_rds);

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

- Einzelheiten zur API finden Sie unter API-Referenz zum [Ändern DBCluster ParameterGroup](#) im AWS SDK für Rust.

Auto Scaling Scaling-Beispiele mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe des AWS SDK für Rust mit Auto Scaling Aktionen ausführen und gängige Szenarien implementieren.

Bei Grundlagen handelt es sich um Code-Beispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo Auto Scaling

Die folgenden Codebeispiele zeigen, wie Sie mit Auto Scaling beginnen können.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [DescribeAutoScalingGroups](#) in der API-Referenz zum AWS SDK für Rust.

Themen

- [Grundlagen](#)
- [Aktionen](#)

Grundlagen

Erlernen der Grundlagen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie eine Amazon EC2 Auto Scaling Scaling-Gruppe mit einer Startvorlage und Availability Zones und erhalten Sie Informationen über laufende Instances.
- Aktivieren Sie die Erfassung von CloudWatch Amazon-Metriken.
- Aktualisieren Sie die gewünschte Kapazität der Gruppe und warten Sie, bis eine Instance gestartet wird.
- Beenden Sie eine Instanz in der Gruppe.
- Listet Skalierungsaktivitäten auf, die als Reaktion auf Benutzeranfragen und Kapazitätsänderungen erfolgen.
- Holen Sie sich Statistiken für CloudWatch Metriken und bereinigen Sie dann Ressourcen.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

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

[dependencies]
aws-config = { version = "1.0.1", features = ["behavior-version-latest"] }
aws-sdk-autoscaling = { version = "1.3.0" }
aws-sdk-ec2 = { version = "1.3.0" }
aws-types = { version = "1.0.1" }
```

```
tokio = { version = "1.20.1", features = ["full"] }
clap = { version = "4.4", features = ["derive"] }
tracing-subscriber = { version = "0.3.15", features = ["env-filter"] }
anyhow = "1.0.75"
tracing = "0.1.37"
tokio-stream = "0.1.14"

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

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

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

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

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

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

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

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

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

    let mut warnings = Warnings::default();

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

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

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

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

    // 5. UpdateAutoScalingGroup: update max size to 3.
```

```
let scale_max_size = scenario.scale_max_size(3).await;
if let Err(err) = scale_max_size {
    warnings.push("There was a problem scaling max size", err);
}

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

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

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

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

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

// 9. TerminateInstanceInAutoScalingGroup: terminate one of the instances in the
group.
let terminate_some_instance = scenario.terminate_some_instance().await;
if let Err(err) = terminate_some_instance {
```

```
        warnings.push("There was a problem replacing an instance", err);
    }

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

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

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

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

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

    // 11. DisableMetricsCollection
    let scale_group = scenario.scale_group_to_zero().await;
```

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

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

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

pub mod scenario;

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

use anyhow::anyhow;
use aws_config::SdkConfig;
use aws_sdk_autoscaling::{
    error::{DisplayErrorContext, ProvideErrorMetadata},
    types::{Activity, AutoScalingGroup, LaunchTemplateSpecification},
};
use aws_sdk_ec2::types::RequestLaunchTemplateData;
use tracing::trace;
```



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

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

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

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

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



```

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

    Ok(())
}

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

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

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

```

```
    }
}

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

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

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

impl Error for ScenarioError {
    // While `Error` can capture `source` information about the underlying error,
    // for this example
    // the ScenarioError captures the underlying information in MetadataError and
    // treats it as a
    // single Error from this Crate. In other contexts, it may be appropriate to
    // model the error
    // as including the SdkError as its source.
}

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

impl AutoScalingScenario {
```

```

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

    let auto_scaling_group_name = String::from(AUTOSCALING_GROUP_NAME);

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

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

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

    let launch_template_arn = match create_launch_template.launch_template {
        Some(launch_template) =>
launch_template.launch_template_id.unwrap_or_default(),
        None => {

```

```

        // Try to delete the launch template
        let _ = ec2
            .delete_launch_template()
            .launch_template_name(LAUNCH_TEMPLATE_NAME)
            .send()
            .await;
        return Err(vec![ScenarioError::with("Failed to load launch
template")]));
    }
};

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

    if let Err(err) = autoscaling
        .delete_auto_scaling_group()
        .auto_scaling_group_name(auto_scaling_group_name.as_str())
        .send()
        .await
    {
        errs.push(ScenarioError::new(

```

```
        "Failed to clean up autoscaling group",
        &err,
    ));
}

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

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

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

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

```

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

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

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

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

```



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

```

}

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

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

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

```

```
        .items()
        .send()
        .collect::()
        .await
        .map_err(|e| {
            anyhow!(
                "There was an error retrieving scaling activities: {}",
                DisplayErrorContext(&e)
            )
        });

    AutoScalingScenarioDescription {
        group,
        instances,
        activities,
    }
}

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

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

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

    if auto_scaling_group.is_none() {
```

```

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

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

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

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

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

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

```

```

        group = self.get_group().await?;
        count = count_group_instances(&group);
    }

    Ok(())
}

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

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

pub async fn scale_min_size(&self, size: i32) -> Result<(), ScenarioError> {
    let update_group = self
        .autoscaling
        .update_auto_scaling_group()
        .auto_scaling_group_name(self.auto_scaling_group_name.clone())
        .min_size(size)

```

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

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

pub async fn scale_desired_capacity(&self, capacity: i32) -> Result<(),
ScenarioError> {
    // 7. SetDesiredCapacity: set desired capacity to 2.
    // Wait for a second instance to launch.
    let update_group = self
        .autoscaling
        .set_desired_capacity()
        .auto_scaling_group_name(self.auto_scaling_group_name.clone())
        .desired_capacity(capacity)
        .send()
        .await;
    if let Err(err) = update_group {
        return Err(ScenarioError::new(
            format!("Failed to update group to desired capacity
({capacity}))").as_str(),
```

```
        &err,
    ));
}
Ok(())
}

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

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

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

    Ok(())
}
```

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

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

- Weitere API-Informationen finden Sie in den folgenden Themen der API-Referenz zu AWS - SDK für Rust.
 - [CreateAutoScalingGroup](#)
 - [DeleteAutoScalingGroup](#)
 - [DescribeAutoScalingGroups](#)

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

Aktionen

CreateAutoScalingGroup

Das folgende Codebeispiel zeigt, wie man es benutzt `CreateAutoScalingGroup`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [CreateAutoScalingGroup](#) in der API-Referenz zum AWS SDK für Rust.

DeleteAutoScalingGroup

Das folgende Codebeispiel zeigt die Verwendung `DeleteAutoScalingGroup`.

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [DeleteAutoScalingGroup](#) in der API-Referenz zum AWS SDK für Rust.

DescribeAutoScalingGroups

Das folgende Codebeispiel zeigt die Verwendung `DescribeAutoScalingGroups`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [DescribeAutoScalingGroups](#) in der API-Referenz zum AWS SDK für Rust.

DescribeAutoScalingInstances

Das folgende Codebeispiel zeigt die Verwendung `DescribeAutoScalingInstances`.

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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))
    }
}
```

- Einzelheiten zur API finden Sie [DescribeAutoScalingInstances](#) in der API-Referenz zum AWS SDK für Rust.

DescribeScalingActivities

Das folgende Codebeispiel zeigt die Verwendung `DescribeScalingActivities`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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,
    }
}

```

- Einzelheiten zur API finden Sie [DescribeScalingActivities](#) in der API-Referenz zum AWS SDK für Rust.

DisableMetricsCollection

Das folgende Codebeispiel zeigt die Verwendung `DisableMetricsCollection`.

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// 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;
```

- Einzelheiten zur API finden Sie [DisableMetricsCollection](#) in der API-Referenz zum AWS SDK für Rust.

EnableMetricsCollection

Das folgende Codebeispiel zeigt die Verwendung `EnableMetricsCollection`.

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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;

```

- Einzelheiten zur API finden Sie [EnableMetricsCollection](#) in der API-Referenz zum AWS SDK für Rust.

SetDesiredCapacity

Das folgende Codebeispiel zeigt die Verwendung `SetDesiredCapacity`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

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(())
}
```

- Einzelheiten zur API finden Sie [SetDesiredCapacity](#) in der API-Referenz zum AWS SDK für Rust.

TerminateInstanceInAutoScalingGroup

Das folgende Codebeispiel zeigt die Verwendung `TerminateInstanceInAutoScalingGroup`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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())
}
```

- Einzelheiten zur API finden Sie [TerminateInstanceInAutoScalingGroup](#) in der API-Referenz zum AWS SDK für Rust.

UpdateAutoScalingGroup

Das folgende Codebeispiel zeigt die Verwendung `UpdateAutoScalingGroup`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [UpdateAutoScalingGroup](#) in der API-Referenz zum AWS SDK für Rust.

Amazon Bedrock Runtime-Beispiele mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe des AWS SDK für Rust mit Amazon Bedrock Runtime Aktionen ausführen und allgemeine Szenarien implementieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Themen

- [Anthropic Claude](#)

Anthropic Claude

Converse

Das folgende Codebeispiel zeigt, wie Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Anthropic Claude senden.

SDK für Rust

Note

Es gibt noch mehr dazu. [GitHub](#) Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Senden Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Anthropic Claude.

```
#[tokio::main]
async fn main() -> Result<(), BedrockConverseError> {
    tracing_subscriber::fmt::init();
    let sdk_config = aws_config::defaults(BehaviorVersion::latest())
        .region(CLAUDE_REGION)
        .load()
        .await;
    let client = Client::new(&sdk_config);

    let response = client
        .converse()
        .model_id(MODEL_ID)
        .messages(
            Message::builder()
                .role(ConversationRole::User)
                .content(ContentBlock::Text(USER_MESSAGE.to_string()))
                .build()
                .map_err(|_| "failed to build message"?),
```

```

    )
    .send()
    .await;

    match response {
        Ok(output) => {
            let text = get_converse_output_text(output)?;
            println!("{}", text);
            Ok(())
        }
        Err(e) => Err(e
            .as_service_error()
            .map(BedrockConverseError::from)
            .unwrap_or_else(|| BedrockConverseError("Unknown service
error".into()))),
    }
}

fn get_converse_output_text(output: ConverseOutput) -> Result<String,
BedrockConverseError> {
    let text = output
        .output()
        .ok_or("no output")?
        .as_message()
        .map_err(|_| "output not a message")?
        .content()
        .first()
        .ok_or("no content in message")?
        .as_text()
        .map_err(|_| "content is not text")?
        .to_string();
    Ok(text)
}

```

Verwenden Sie Anweisungen, das Error-Hilfsprogramm und Konstanten.

```

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",
        })
    }
}
}

```

- Einzelheiten zur API finden Sie unter [Converse](#) in der API-Referenz zum AWS SDK für Rust.

ConverseStream

Das folgende Codebeispiel zeigt, wie Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Anthropic Claude senden und den Antwortstream in Echtzeit verarbeiten.

SDK für Rust

 Note

Es gibt noch mehr dazu. GitHub Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Senden Sie mithilfe der Bedrock-API eine Textnachricht an Anthropic Claude und streamen Sie Antwort-Token. `ConverseStream`

```
#[tokio::main]
async fn main() -> Result<(), BedrockConverseStreamError> {
    tracing_subscriber::fmt::init();
    let sdk_config = aws_config::defaults(BehaviorVersion::latest())
        .region(CLAUDE_REGION)
        .load()
        .await;
    let client = Client::new(&sdk_config);

    let response = client
        .converse_stream()
        .model_id(MODEL_ID)
        .messages(
            Message::builder()
                .role(ConversationRole::User)
                .content(ContentBlock::Text(USER_MESSAGE.to_string()))
                .build()
                .map_err(|_| "failed to build message"?),
        )
        .send()
        .await;

    let mut stream = match response {
        Ok(output) => Ok(output.stream),
        Err(e) => Err(BedrockConverseStreamError::from(
            e.as_service_error().unwrap(),
        )),
    };

    loop {
        let token = stream.recv().await;
```

```

        match token {
            Ok(Some(text)) => {
                let next = get_converse_output_text(text)?;
                print!("{}", next);
                Ok(())
            }
            Ok(None) => break,
            Err(e) => Err(e
                .as_service_error()
                .map(BedrockConverseStreamError::from)
                .unwrap_or(BedrockConverseStreamError(
                    "Unknown error receiving stream".into(),
                ))),
        }?
    }

    println!();

    Ok(())
}

fn get_converse_output_text(
    output: ConverseStreamOutputType,
) -> Result<String, BedrockConverseStreamError> {
    Ok(match output {
        ConverseStreamOutputType::ContentBlockDelta(event) => match event.delta() {
            Some(delta) => delta.as_text().cloned().unwrap_or_else(|_| "".into()),
            None => "".into(),
        },
        _ => "".into(),
    })
}

```

Verwenden Sie Anweisungen, das Error-Hilfsprogramm und Konstanten.

```

use aws_config::BehaviorVersion;
use aws_sdk_bedrockruntime::{
    error::ProvideErrorMetadata,
    operation::converse_stream::ConverseStreamError,
    types::{
        error::ConverseStreamOutputError, ContentBlock, ConversationRole,

```



```

        ConverseStreamOutput as ConverseStreamOutputType, Message,
    },
    Client,
};

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

// Start a conversation with the user message.
const USER_MESSAGE: &str = "Describe the purpose of a 'hello world' program in one
line.";

#[derive(Debug)]
struct BedrockConverseStreamError(String);
impl std::fmt::Display for BedrockConverseStreamError {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(f, "Can't invoke '{}'. Reason: {}", MODEL_ID, self.0)
    }
}
impl std::error::Error for BedrockConverseStreamError {}
impl From<&str> for BedrockConverseStreamError {
    fn from(value: &str) -> Self {
        BedrockConverseStreamError(value.into())
    }
}

impl From<&ConverseStreamError> for BedrockConverseStreamError {
    fn from(value: &ConverseStreamError) -> Self {
        BedrockConverseStreamError(
            match value {
                ConverseStreamError::ModelTimeoutException(_) => "Model took too
long",
                ConverseStreamError::ModelNotReadyException(_) => "Model is not
ready",
                _ => "Unknown",
            }
            .into(),
        )
    }
}

impl From<&ConverseStreamOutputError> for BedrockConverseStreamError {
    fn from(value: &ConverseStreamOutputError) -> Self {

```

```
        match value {
            ConverseStreamOutputError::ValidationException(ve) =>
BedrockConverseStreamError(
                ve.message().unwrap_or("Unknown ValidationException").into(),
            ),
            ConverseStreamOutputError::ThrottlingException(te) =>
BedrockConverseStreamError(
                te.message().unwrap_or("Unknown ThrottlingException").into(),
            ),
            value => BedrockConverseStreamError(
                value
                    .message()
                    .unwrap_or("Unknown StreamOutput exception")
                    .into(),
            ),
        }
    }
}
```

- Einzelheiten zur API finden Sie [ConverseStream](#) in der API-Referenz zum AWS SDK für Rust.

Szenario: Verwendung des Tools mit der Converse-API

Das folgende Codebeispiel zeigt, wie eine typische Interaktion zwischen einer Anwendung, einem generativen KI-Modell und verbundenen Tools aufgebaut oder APIs Interaktionen zwischen der KI und der Außenwelt vermittelt werden. Es verwendet das Beispiel der Verbindung einer externen Wetter-API mit dem KI-Modell, sodass Wetterinformationen in Echtzeit auf der Grundlage von Benutzereingaben bereitgestellt werden können.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Das primäre Szenario und die Logik für die Demo. Dies orchestriert die Konversation zwischen dem Benutzer, der Amazon Bedrock Converse API und einem Wetter-Tool.

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

impl ToolUseScenario {
    fn new(client: Client) -> Self {
        let system_prompt = SystemContentBlock::Text(SYSTEM_PROMPT.into());
        let tool_config = ToolConfiguration::builder()
            .tools(Tool::ToolSpec(
                ToolSpecification::builder()
                    .name(TOOL_NAME)
                    .description(TOOL_DESCRIPTION)
                    .input_schema(ToolInputSchema::Json(make_tool_schema()))
                    .build()
                    .unwrap(),
            ))
            .build()
            .unwrap();

        ToolUseScenario {
            client,
            conversation: vec![],
            system_prompt,
            tool_config,
        }
    }

    async fn run(&mut self) -> Result<(), ToolUseScenarioError> {
        loop {
            let input = get_input().await?;
            if input.is_none() {
                break;
            }

            let message = Message::builder()
                .role(User)
                .content(ContentBlock::Text(input.unwrap()))
        }
    }
}
```

```

        .build()
        .map_err(ToolUseScenarioError::from)?;
self.conversation.push(message);

let response = self.send_to_bedrock().await?;

self.process_model_response(response).await?;
}

Ok(())
}

async fn send_to_bedrock(&mut self) -> Result<ConverseOutput,
ToolUseScenarioError> {
    debug!("Sending conversation to bedrock");
    self.client
        .converse()
        .model_id(MODEL_ID)
        .set_messages(Some(self.conversation.clone()))
        .system(self.system_prompt.clone())
        .tool_config(self.tool_config.clone())
        .send()
        .await
        .map_err(ToolUseScenarioError::from)
}

async fn process_model_response(
    &mut self,
    mut response: ConverseOutput,
) -> Result<(), ToolUseScenarioError> {
    let mut iteration = 0;

    while iteration < MAX_RECURSIONS {
        iteration += 1;
        let message = if let Some(ref output) = response.output {
            if output.is_message() {
                Ok(output.as_message().unwrap().clone())
            } else {
                Err(ToolUseScenarioError(
                    "Converse Output is not a message".into(),
                ))
            }
        } else {
            Err(ToolUseScenarioError("Missing Converse Output".into()))
        }
    }
}

```

```
    }?;

    self.conversation.push(message.clone());

    match response.stop_reason {
        StopReason::ToolUse => {
            response = self.handle_tool_use(&message).await?;
        }
        StopReason::EndTurn => {
            print_model_response(&message.content[0])?;
            return Ok(());
        }
        _ => (),
    }
}

Err(ToolUseScenarioError(
    "Exceeded MAX_ITERATIONS when calling tools".into(),
))
}

async fn handle_tool_use(
    &mut self,
    message: &Message,
) -> Result<ConverseOutput, ToolUseScenarioError> {
    let mut tool_results: Vec<ContentBlock> = vec![];

    for block in &message.content {
        match block {
            ContentBlock::Text(_) => print_model_response(block)?,
            ContentBlock::ToolUse(tool) => {
                let tool_response = self.invoke_tool(tool).await?;
                tool_results.push(ContentBlock::ToolResult(tool_response.1));
            }
            _ => (),
        };
    }

    let message = Message::builder()
        .role(User)
        .set_content(Some(tool_results))
        .build()?;
    self.conversation.push(message);
}
```

```

        self.send_to_bedrock().await
    }

    async fn invoke_tool(
        &mut self,
        tool: &ToolUseBlock,
    ) -> Result<InvokeToolResult, ToolUseScenarioError> {
        match tool.name() {
            TOOL_NAME => {
                println!(
                    "\x1b[0;90mExecuting tool: {TOOL_NAME} with input: {:?}...
\x1b[0m",
                    tool.input()
                );
                let content = fetch_weather_data(tool).await?;
                println!(
                    "\x1b[0;90mTool responded with {:?}\x1b[0m",
                    content.content()
                );
                Ok(InvokeToolResult(tool.tool_use_id.clone(), content))
            }
            _ => Err(ToolUseScenarioError(format!(
                "The requested tool with name {} does not exist",
                tool.name()
            ))),
        }
    }
}

#[tokio::main]
async fn main() {
    tracing_subscriber::fmt::init();
    let sdk_config = aws_config::defaults(BehaviorVersion::latest())
        .region(CLAUDE_REGION)
        .load()
        .await;
    let client = Client::new(&sdk_config);

    let mut scenario = ToolUseScenario::new(client);

    header();
    if let Err(err) = scenario.run().await {
        println!("There was an error running the scenario! {}", err.0)
    }
}

```

```
    footer();  
}
```

Das von der Demo verwendete Wetter-Tool. Dieses Skript definiert die Werkzeugspezifikation und implementiert die Logik zum Abrufen von Wetterdaten mithilfe der 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()?)
}

```

Hilfsprogramme zum Drucken der Nachrichteninhaltsblöcke.

```

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:?})"
        )))
    }
}

```

Verwenden Sie Anweisungen, das Error-Hilfsprogramm und Konstanten.

```

use std::{collections::HashMap, io::stdin};

use aws_config::BehaviorVersion;
use aws_sdk_bedrockruntime::{
    error::{BuildError, SdkError},

```



```

    operation::converse::{ConverseError, ConverseOutput},
    types::{
        ContentBlock, ConversationRole::User, Message, StopReason,
        SystemContentBlock, Tool,
        ToolConfiguration, ToolInputSchema, ToolResultBlock, ToolResultContentBlock,
        ToolSpecification, ToolUseBlock,
    },
    Client,
};
use aws_smithy_runtime_api::http::Response;
use aws_smithy_types::Document;
use tracing::debug;

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

const SYSTEM_PROMPT: &str = "You are a weather assistant that provides current
    weather data for user-specified locations using only
    the Weather_Tool, which expects latitude and longitude. Infer the coordinates from
    the location yourself.
    If the user provides coordinates, infer the approximate location and refer to it in
    your response.
    To use the tool, you strictly apply the provided tool specification.

    - Explain your step-by-step process, and give brief updates before each step.
    - Only use the Weather_Tool for data. Never guess or make up information.
    - Repeat the tool use for subsequent requests if necessary.
    - If the tool errors, apologize, explain weather is unavailable, and suggest other
    options.
    - Report temperatures in °C (°F) and wind in km/h (mph). Keep weather reports
    concise. Sparingly use
    emojis where appropriate.
    - Only respond to weather queries. Remind off-topic users of your purpose.
    - Never claim to search online, access external data, or use tools besides
    Weather_Tool.
    - Complete the entire process until you have all required data before sending the
    complete response.
";

// The maximum number of recursive calls allowed in the tool_use_demo function.
// This helps prevent infinite loops and potential performance issues.
const MAX_RECURSIONS: i8 = 5;

```

```

const TOOL_NAME: &str = "Weather_Tool";
const TOOL_DESCRIPTION: &str =
    "Get the current weather for a given location, based on its WGS84 coordinates.";
fn make_tool_schema() -> Document {
    Document::Object(HashMap::<String, Document>::from([
        ("type".into(), Document::String("object".into())),
        (
            "properties".into(),
            Document::Object(HashMap::from([
                (
                    "latitude".into(),
                    Document::Object(HashMap::from([
                        ("type".into(), Document::String("string".into())),
                        (
                            "description".into(),
                            Document::String("Geographical WGS84 latitude of the
location.".into()),
                        ),
                    ])),
                ),
                (
                    "longitude".into(),
                    Document::Object(HashMap::from([
                        ("type".into(), Document::String("string".into())),
                        (
                            "description".into(),
                            Document::String(
                                "Geographical WGS84 longitude of the
location.".into()),
                        ),
                    ])),
                ),
            ])),
        ),
        (
            "required".into(),
            Document::Array(vec![
                Document::String("latitude".into()),
                Document::String("longitude".into()),
            ]),
        ),
    ]))
}

```

```

#[derive(Debug)]
struct ToolUseScenarioError(String);
impl std::fmt::Display for ToolUseScenarioError {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(f, "Tool use error with '{}'. Reason: {}", MODEL_ID, self.0)
    }
}
impl From<&str> for ToolUseScenarioError {
    fn from(value: &str) -> Self {
        ToolUseScenarioError(value.into())
    }
}
impl From<BuildError> for ToolUseScenarioError {
    fn from(value: BuildError) -> Self {
        ToolUseScenarioError(value.to_string().clone())
    }
}
impl From<SdkError<ConverseError, Response>> for ToolUseScenarioError {
    fn from(value: SdkError<ConverseError, Response>) -> Self {
        ToolUseScenarioError(match value.as_service_error() {
            Some(value) => value.meta().message().unwrap_or("Unknown").into(),
            None => "Unknown".into(),
        })
    }
}
}

```

- Einzelheiten zur API finden Sie unter [Converse](#) in der API-Referenz zum AWS SDK für Rust.

Beispiele für Amazon Cognito Identity Provider mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe des AWS SDK für Rust mit Amazon Cognito Identity Provider Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

ListUserPools

Das folgende Codebeispiel zeigt die Verwendung `ListUserPools`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [ListUserPools](#) in der API-Referenz zum AWS SDK für Rust.

Amazon Cognito Sync-Beispiele mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe des AWS SDK für Rust mit Amazon Cognito Sync Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

ListIdentityPoolUsage

Das folgende Codebeispiel zeigt die Verwendung `ListIdentityPoolUsage`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [ListIdentityPoolUsage](#) in der API-Referenz zum AWS SDK für Rust.

Firehose-Beispiele mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe des AWS SDK für Rust mit Firehose Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

PutRecordBatch

Das folgende Codebeispiel zeigt die Verwendung `PutRecordBatch`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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
}
```

- Einzelheiten zur API finden Sie [PutRecordBatch](#) in der API-Referenz zum AWS SDK für Rust.

Amazon DocumentDB DocumentDB-Beispiele mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe des AWS SDK für Rust mit Amazon DocumentDB Aktionen ausführen und allgemeine Szenarien implementieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Themen

- [Serverless-Beispiele](#)

Serverless-Beispiele

Aufrufen einer Lambda-Funktion über einen Amazon DocumentDB-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Datensätzen aus einem DocumentDB-Änderungsstream ausgelöst wird. Die Funktion ruft die DocumentDB-Nutzdaten ab und protokolliert den Inhalt des Datensatzes.

SDK für Rust

Note

Es gibt noch mehr dazu. [GitHub](#) Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Nutzen eines Amazon DocumentDB-Ereignisses mit Lambda unter Verwendung von 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-Beispiele mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe des AWS SDK für Rust mit DynamoDB Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Szenarien sind Code-Beispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Services aufrufen oder mit anderen AWS-Services kombinieren.

AWS Community-Beiträge sind Beispiele, die von mehreren Teams erstellt wurden und verwaltet werden. AWS Um Feedback zu geben, verwende den Mechanismus, der in den verlinkten Repositorien zur Verfügung steht.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)
- [Szenarien](#)
- [Serverless-Beispiele](#)
- [AWS Beiträge der Gemeinschaft](#)

Aktionen

CreateTable

Das folgende Codebeispiel zeigt die Verwendung `CreateTable`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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))
    }
}
```

- Einzelheiten zur API finden Sie [CreateTable](#) in der API-Referenz zum AWS SDK für Rust.

DeleteItem

Das folgende Codebeispiel zeigt die Verwendung `DeleteItem`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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)),
    }
}

```

- Einzelheiten zur API finden Sie [DeleteItem](#) in der API-Referenz zum AWS SDK für Rust.

DeleteTable

Das folgende Codebeispiel zeigt die Verwendung `DeleteTable`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

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())),
    }
}

```

- Einzelheiten zur API finden Sie [DeleteTable](#) in der API-Referenz zum AWS SDK für Rust.

ListTables

Das folgende Codebeispiel zeigt die Verwendung `ListTables`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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::
```

Finden Sie heraus, ob eine Tabelle vorhanden ist.

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

- Einzelheiten zur API finden Sie [ListTables](#) in der API-Referenz zum AWS SDK für Rust.

PutItem

Das folgende Codebeispiel zeigt die Verwendung `PutItem`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
pub async fn add_item(client: &Client, item: Item, table: &String) ->
Result<ItemOut, Error> {
    let user_av = AttributeValue::S(item.username);
    let type_av = AttributeValue::S(item.p_type);
    let age_av = AttributeValue::S(item.age);
    let first_av = AttributeValue::S(item.first);
    let last_av = AttributeValue::S(item.last);

    let request = client
        .put_item()
        .table_name(table)
        .item("username", user_av)
        .item("account_type", type_av)
        .item("age", age_av)
        .item("first_name", first_av)
        .item("last_name", last_av);

    println!("Executing request [{request:?}] to add item...");

    let resp = request.send().await?;

    let attributes = resp.attributes().unwrap();

    let username = attributes.get("username").cloned();
    let first_name = attributes.get("first_name").cloned();
    let last_name = attributes.get("last_name").cloned();
    let age = attributes.get("age").cloned();
    let p_type = attributes.get("p_type").cloned();

    println!(
        "Added user {username}, {first_name} {last_name}, age {age} as {p_type} user",
```

```

        username, first_name, last_name, age, p_type
    );

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

```

- Einzelheiten zur API finden Sie [PutItem](#) in der API-Referenz zum AWS SDK für Rust.

Query

Das folgende Codebeispiel zeigt die Verwendung `Query`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Finden Sie die Filme, die im angegebenen Jahr gedreht wurden.

```

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![])
}
}
```

- Weitere API-Informationen finden Sie unter [Query](#) in der API-Referenz zum AWS -SDK für Rust.

Scan

Das folgende Codebeispiel zeigt, wie man es benutzt.

SDK für Rust

Note

Es gibt noch mehr dazu [auf GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())  
}
```

- Weitere API-Informationen finden Sie unter [Scan](#) in der API-Referenz zum AWS -SDK für Rust.

Szenarien

Stellen Sie eine Connect zu einer lokalen Instanz her

Das folgende Codebeispiel zeigt, wie eine Endpunkt-URL überschrieben wird, um eine Verbindung zu einer lokalen Entwicklungsbereitstellung von DynamoDB und einem AWS SDK herzustellen.

Weitere Informationen finden Sie unter [DynamoDB Local](#).

SDK für Rust

Note

Weitere Informationen finden Sie unter. GitHub Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

Erstellen einer Serverless-Anwendung zur Verwaltung von Fotos

Das folgende Codebeispiel zeigt, wie eine Serverless-Anwendung erstellt wird, mit der Benutzer Fotos mithilfe von Labels erstellen können.

SDK für Rust

Zeigt, wie eine Anwendung zur Verwaltung von Fotobeständen entwickelt wird, die mithilfe von Amazon Rekognition Labels in Bildern erkennt und sie für einen späteren Abruf speichert.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

Einen tiefen Einblick in den Ursprung dieses Beispiels finden Sie im Beitrag in der [AWS - Community](#).

In diesem Beispiel verwendete Dienste

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

Abfragen einer Tabelle mit PartiQL

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Abrufen eines Elementes durch Ausführen einer SELECT-Anweisung.
- Hinzufügen eines Elementes durch Ausführung einer INSERT-Anweisung.
- Aktualisieren eines Elementes durch Ausführung einer UPDATE-Anweisung.
- Löschen eines Elementes durch Ausführung einer DELETE-Anweisung.

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
async fn make_table(
    client: &Client,
    table: &str,
    key: &str,
) -> Result<(), SdkError<CreateTableError>> {
    let ad = AttributeDefinition::builder()
        .attribute_name(key)
        .attribute_type(ScalarAttributeType::S)
        .build()
        .expect("creating AttributeDefinition");

    let ks = KeySchemaElement::builder()
        .attribute_name(key)
        .key_type(KeyType::Hash)
        .build()
        .expect("creating KeySchemaElement");

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

```

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

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

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

```

```

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

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

    println!("Deleted item.");

    Ok(())
}

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

```

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

- Einzelheiten zur API finden Sie [ExecuteStatement](#) in der API-Referenz zum AWS SDK für Rust.

EXIF- und andere Bildinformationen speichern

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Rufen Sie EXIF-Informationen aus einer JPG-, JPEG- oder PNG-Datei ab.
- Laden Sie die Bilddatei in einen Amazon-S3-Bucket hoch.
- Verwenden Sie Amazon Rekognition, um die drei wichtigsten Attribute (Labels) in der Datei zu identifizieren.
- Fügen Sie die EXIF- und Label-Informationen einer Amazon-DynamoDB-Tabelle in der Region hinzu.

SDK für Rust

Rufen Sie EXIF-Informationen aus einer JPG-, JPEG- oder PNG-Datei ab, laden Sie die Bilddatei in einen Amazon-S3-Bucket hoch und identifizieren Sie mit Amazon Rekognition die drei wichtigsten Attribute (Labels in Amazon Rekognition) in der Datei. Fügen Sie die EXIF- und Labelinformationen dann einer Amazon-DynamoDB-Tabelle in der Region hinzu.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- DynamoDB
- Amazon Rekognition
- Amazon S3

Serverless-Beispiele

Aufrufen einer Lambda-Funktion über einen DynamoDB-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Datensätzen aus einem DynamoDB-Stream ausgelöst wird. Die Funktion ruft die DynamoDB-Nutzdaten ab und protokolliert den Inhalt des Datensatzes.

SDK für Rust

Note

Es gibt noch mehr dazu. [GitHub](#) Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Nutzen eines DynamoDB-Ereignisses mit Lambda unter Verwendung von 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(())
}
```

Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem DynamoDB-Auslöser

Das folgende Codebeispiel zeigt, wie eine partielle Batch-Antwort für Lambda-Funktionen implementiert wird, die Ereignisse aus einem DynamoDB-Stream empfangen. Die Funktion meldet die Batch-Elementfehler in der Antwort und signalisiert Lambda, diese Nachrichten später erneut zu versuchen.

SDK für Rust

 Note

Es gibt noch mehr dazu. [GitHub](#) Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Melden von DynamoDB-Batchelementfehlern mit Lambda unter Verwendung von 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...
async fn function_handler(event: LambdaEvent<Event>) ->
Result<DynamoDbEventResponse, Error> {
    let mut response = DynamoDbEventResponse {
        batch_item_failures: vec![],
    };

    let records = &event.payload.records;

    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(response);
    }

    for record in records {
        tracing::info!("EventId: {}", record.event_id);
    }
}
```

```
// Couldn't find a sequence number
if record.change.sequence_number.is_none() {
    response.batch_item_failures.push(DynamoDbBatchItemFailure {
        item_identifier: Some("").to_string(),
    });
    return Ok(response);
}

// Process your record here...
if process_record(record).is_err() {
    response.batch_item_failures.push(DynamoDbBatchItemFailure {
        item_identifier: record.change.sequence_number.clone(),
    });
    /* Since we are working with streams, we can return the failed item
immediately.
    Lambda will immediately begin to retry processing from this failed item
onwards. */
    return Ok(response);
}

tracing::info!("Successfully processed {} record(s)", records.len());

Ok(response)
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

AWS Beiträge der Gemeinschaft

Erstellen und testen Sie eine serverlose Anwendung

Das folgende Codebeispiel zeigt, wie eine serverlose Anwendung mithilfe von API Gateway mit Lambda und DynamoDB erstellt und getestet wird.

SDK für Rust

Zeigt, wie eine serverlose Anwendung, die aus einem API Gateway mit Lambda und DynamoDB besteht, mithilfe des Rust-SDK erstellt und getestet wird.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter. [GitHub](#)

In diesem Beispiel verwendete Dienste

- API Gateway
- DynamoDB
- Lambda

Amazon EBS-Beispiele mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe des AWS SDK für Rust mit Amazon EBS Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

CompleteSnapshot

Das folgende Codebeispiel zeigt die Verwendung `CompleteSnapshot`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [CompleteSnapshot](#) in der API-Referenz zum AWS SDK für Rust.

PutSnapshotBlock

Das folgende Codebeispiel zeigt, wie man es benutzt `PutSnapshotBlock`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [PutSnapshotBlock](#) in der API-Referenz zum AWS SDK für Rust.

StartSnapshot

Das folgende Codebeispiel zeigt, wie man es benutzt `StartSnapshot`.

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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())
}
```

- Einzelheiten zur API finden Sie [StartSnapshot](#) in der API-Referenz zum AWS SDK für Rust.

EC2 Amazon-Beispiele mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe des AWS SDK für Rust mit Amazon Aktionen ausführen und allgemeine Szenarien implementieren EC2.

Bei Grundlagen handelt es sich um Code-Beispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo Amazon EC2

Die folgenden Codebeispiele zeigen, wie Sie mit Amazon beginnen können EC2.

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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


- Einzelheiten zur API finden Sie [DescribeSecurityGroups](#) in der API-Referenz zum AWS SDK für Rust.

Themen

- [Grundlagen](#)
- [Aktionen](#)

Grundlagen

Erlernen der Grundlagen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie ein Schlüsselpaar und eine Sicherheitsgruppe.
- Wählen Sie ein Amazon Machine Image (AMI) und einen kompatiblen Instance-Typ aus und erstellen Sie anschließend eine Instance.
- Halten Sie die Instance an und starten Sie sie neu.
- Verknüpfen einer Elastic-IP-Adresse mit der Instance.
- Stellen Sie über SSH eine Verbindung zu Ihrer Instance her und bereinigen Sie dann die Ressourcen.

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Die EC2 InstanceScenario Implementierung enthält Logik, um das Beispiel als Ganzes auszuführen.

```
//! Scenario that uses the AWS SDK for Rust (the SDK) with Amazon Elastic Compute
    Cloud
    //! (Amazon EC2) to do the following:
    //!
```

```
//! * Create a key pair that is used to secure SSH communication between your
    computer and
    an EC2 instance.
    * Create a security group that acts as a virtual firewall for your EC2 instances
    to
    control incoming and outgoing traffic.
    * Find an Amazon Machine Image (AMI) and a compatible instance type.
    * Create an instance that is created from the instance type and AMI you select,
    and
    is configured to use the security group and key pair created in this example.
    * Stop and restart the instance.
    * Create an Elastic IP address and associate it as a consistent IP address for
    your instance.
    * Connect to your instance with SSH, using both its public IP address and your
    Elastic IP
    address.
    * Clean up all of the resources created by this example.

use std::net::Ipv4Addr;

use crate::{
    ec2::{EC2Error, EC2},
    getting_started::{key_pair::KeyPairManager, util::Util},
    ssm::SSM,
};
use aws_sdk_ssm::types::Parameter;

use super::{
    elastic_ip::ElasticIpManager, instance::InstanceManager,
    security_group::SecurityGroupManager,
    util::ScenarioImage,
};

pub struct Ec2InstanceScenario {
    ec2: EC2,
    ssm: SSM,
    util: Util,
    key_pair_manager: KeyPairManager,
    security_group_manager: SecurityGroupManager,
    instance_manager: InstanceManager,
    elastic_ip_manager: ElasticIpManager,
}

impl Ec2InstanceScenario {
```

```

pub fn new(ec2: EC2, ssm: SSM, util: Util) -> Self {
    Ec2InstanceScenario {
        ec2,
        ssm,
        util,
        key_pair_manager: Default::default(),
        security_group_manager: Default::default(),
        instance_manager: Default::default(),
        elastic_ip_manager: Default::default(),
    }
}

pub async fn run(&mut self) -> Result<(), EC2Error> {
    self.create_and_list_key_pairs().await?;
    self.create_security_group().await?;
    self.create_instance().await?;
    self.stop_and_start_instance().await?;
    self.associate_elastic_ip().await?;
    self.stop_and_start_instance().await?;
    Ok(())
}

/// 1. Creates an RSA key pair and saves its private key data as a .pem file in
secure
/// temporary storage. The private key data is deleted after the example
completes.
/// 2. Optionally, lists the first five key pairs for the current account.
pub async fn create_and_list_key_pairs(&mut self) -> Result<(), EC2Error> {
    println!( "Let's create an RSA key pair that you can be use to securely
connect to your EC2 instance.");

    let key_name = self.util.prompt_key_name()?;

    self.key_pair_manager
        .create(&self.ec2, &self.util, key_name)
        .await?;

    println!(
        "Created a key pair {} and saved the private key to {:?}.",
        self.key_pair_manager
            .key_pair()
            .key_name()
            .ok_or_else(|| EC2Error::new("No key name after creating key")),
        self.key_pair_manager

```

```

        .key_file_path()
        .ok_or_else(|| EC2Error::new("No key file after creating key"))?
    );

    if self.util.should_list_key_pairs()? {
        for pair in self.key_pair_manager.list(&self.ec2).await? {
            println!(
                "Found {:?} key {} with fingerprint:\t{:?}",
                pair.key_type(),
                pair.key_name().unwrap_or("Unknown"),
                pair.key_fingerprint()
            );
        }
    }

    Ok(())
}

/// 1. Creates a security group for the default VPC.
/// 2. Adds an inbound rule to allow SSH. The SSH rule allows only
///     inbound traffic from the current computer's public IPv4 address.
/// 3. Displays information about the security group.
///
/// This function uses <http://checkip.amazonaws.com> to get the current public
IP
/// address of the computer that is running the example. This method works in
most
/// cases. However, depending on how your computer connects to the internet, you
/// might have to manually add your public IP address to the security group by
using
/// the AWS Management Console.
pub async fn create_security_group(&mut self) -> Result<(), EC2Error> {
    println!("Let's create a security group to manage access to your
instance.");
    let group_name = self.util.prompt_security_group_name()?;

    self.security_group_manager
        .create(
            &self.ec2,
            &group_name,
            "Security group for example: get started with instances.",
        )
        .await?;
}

```

```

println!(
    "Created security group {} in your default VPC {}.",
    self.security_group_manager.group_name(),
    self.security_group_manager
        .vpc_id()
        .unwrap_or("(unknown vpc)")
);

let check_ip = self.util.do_get("https://checkip.amazonaws.com").await?;
let current_ip_address: Ipv4Addr = check_ip.trim().parse().map_err(|e| {
    EC2Error::new(format!(
        "Failed to convert response {} to IP Address: {e:?}",
        check_ip
    ))
})?;

println!("Your public IP address seems to be {current_ip_address}");
if self.util.should_add_to_security_group() {
    match self
        .security_group_manager
        .authorize_ingress(&self.ec2, current_ip_address)
        .await
    {
        Ok(_) => println!("Security group rules updated"),
        Err(err) => eprintln!("Couldn't update security group rules:
{err:?}"),
    }
}
println!("{}", self.security_group_manager);

Ok(())
}

/// 1. Gets a list of Amazon Linux 2 AMIs from AWS Systems Manager. Specifying
the
///     '/aws/service/ami-amazon-linux-latest' path returns only the latest AMIs.
/// 2. Gets and displays information about the available AMIs and lets you
select one.
/// 3. Gets a list of instance types that are compatible with the selected AMI
and
///     lets you select one.
/// 4. Creates an instance with the previously created key pair and security
group,
///     and the selected AMI and instance type.

```

```

/// 5. Waits for the instance to be running and then displays its information.
pub async fn create_instance(&mut self) -> Result<(), EC2Error> {
    let ami = self.find_image().await?;

    let instance_types = self
        .ec2
        .list_instance_types(&ami.0)
        .await
        .map_err(|e| e.add_message("Could not find instance types"))?;
    println!(
        "There are several instance types that support the {} architecture of
the image.",
        ami.0
            .architecture
            .as_ref()
            .ok_or_else(|| EC2Error::new(format!("Missing architecture in {:?}",
ami.0)))?
    );
    let instance_type = self.util.select_instance_type(instance_types)?;

    println!("Creating your instance and waiting for it to start...");
    self.instance_manager
        .create(
            &self.ec2,
            ami.0
                .image_id()
                .ok_or_else(|| EC2Error::new("Could not find image ID"))?,
            instance_type,
            self.key_pair_manager.key_pair(),
            self.security_group_manager
                .security_group()
                .map(|sg| vec![sg])
                .ok_or_else(|| EC2Error::new("Could not find security group"))?,
        )
        .await
        .map_err(|e| e.add_message("Scenario failed to create instance"))?;

    while let Err(err) = self
        .ec2
        .wait_for_instance_ready(self.instance_manager.instance_id(), None)
        .await
    {
        println!("{}", err);
        if !self.util.should_continue_waiting() {

```

```

        return Err(err);
    }
}

println!("Your instance is ready:\n{}", self.instance_manager);

self.display_ssh_info();

Ok(())
}

async fn find_image(&mut self) -> Result<ScenarioImage, EC2Error> {
    let params: Vec<Parameter> = self
        .ssm
        .list_path("/aws/service/ami-amazon-linux-latest")
        .await
        .map_err(|e| e.add_message("Could not find parameters for available
images"))?
        .into_iter()
        .filter(|param| param.name().is_some_and(|name| name.contains("amzn2")))
        .collect();
    let amzn2_images: Vec<ScenarioImage> = self
        .ec2
        .list_images(params)
        .await
        .map_err(|e| e.add_message("Could not find images"))?
        .into_iter()
        .map(ScenarioImage::from)
        .collect();
    println!("We will now create an instance from an Amazon Linux 2 AMI");
    let ami = self.util.select_scenario_image(amzn2_images)?;
    Ok(ami)
}

// 1. Stops the instance and waits for it to stop.
// 2. Starts the instance and waits for it to start.
// 3. Displays information about the instance.
// 4. Displays an SSH connection string. When an Elastic IP address is
associated
//    with the instance, the IP address stays consistent when the instance stops
//    and starts.
pub async fn stop_and_start_instance(&self) -> Result<(), EC2Error> {
    println!("Let's stop and start your instance to see what changes.");
    println!("Stopping your instance and waiting until it's stopped...");
}

```

```

    self.instance_manager.stop(&self.ec2).await?;
    println!("Your instance is stopped. Restarting...");
    self.instance_manager.start(&self.ec2).await?;
    println!("Your instance is running.");
    println!("{}", self.instance_manager);
    if self.elastic_ip_manager.public_ip() == "0.0.0.0" {
        println!("Every time your instance is restarted, its public IP address
changes.");
    } else {
        println!(
            "Because you have associated an Elastic IP with your instance, you
can connect by using a consistent IP address after the instance restarts."
        );
    }
    self.display_ssh_info();
    Ok(())
}

/// 1. Allocates an Elastic IP address and associates it with the instance.
/// 2. Displays an SSH connection string that uses the Elastic IP address.
async fn associate_elastic_ip(&mut self) -> Result<(), EC2Error> {
    self.elastic_ip_manager.allocate(&self.ec2).await?;
    println!(
        "Allocated static Elastic IP address: {}",
        self.elastic_ip_manager.public_ip()
    );

    self.elastic_ip_manager
        .associate(&self.ec2, self.instance_manager.instance_id())
        .await?;
    println!("Associated your Elastic IP with your instance.");
    println!("You can now use SSH to connect to your instance by using the
Elastic IP.");
    self.display_ssh_info();
    Ok(())
}

/// Displays an SSH connection string that can be used to connect to a running
/// instance.
fn display_ssh_info(&self) {
    let ip_addr = if self.elastic_ip_manager.has_allocation() {
        self.elastic_ip_manager.public_ip()
    } else {
        self.instance_manager.instance_ip()
    }
}

```



```
};
let key_file_path = self.key_pair_manager.key_file_path().unwrap();
println!("To connect, open another command prompt and run the following
command:");
println!("\nssh -i {} ec2-user@[ip_addr]\n", key_file_path.display());
let _ = self.util.enter_to_continue();
}

/// 1. Disassociate and delete the previously created Elastic IP.
/// 2. Terminate the previously created instance.
/// 3. Delete the previously created security group.
/// 4. Delete the previously created key pair.
pub async fn clean_up(self) {
    println!("Let's clean everything up. This example created these
resources:");
    println!(
        "\tKey pair: {}",
        self.key_pair_manager
            .key_pair()
            .key_name()
            .unwrap_or("(unknown key pair)")
    );
    println!(
        "\tSecurity group: {}",
        self.security_group_manager.group_name()
    );
    println!(
        "\tInstance: {}",
        self.instance_manager.instance_display_name()
    );
    if self.util.should_clean_resources() {
        if let Err(err) = self.elastic_ip_manager.remove(&self.ec2).await {
            eprintln!("{err}")
        }
        if let Err(err) = self.instance_manager.delete(&self.ec2).await {
            eprintln!("{err}")
        }
        if let Err(err) = self.security_group_manager.delete(&self.ec2).await {
            eprintln!("{err}");
        }
        if let Err(err) = self.key_pair_manager.delete(&self.ec2,
&self.util).await {
            eprintln!("{err}");
        }
    }
}
```

```

        } else {
            println!("Ok, not cleaning up any resources!");
        }
    }
}

pub async fn run(mut scenario: Ec2InstanceScenario) {
    println!
    ("-----");
    println!(
        "Welcome to the Amazon Elastic Compute Cloud (Amazon EC2) get started with
instances demo."
    );
    println!
    ("-----");

    if let Err(err) = scenario.run().await {
        eprintln!("There was an error running the scenario: {err}")
    }

    println!
    ("-----");

    scenario.clean_up().await;

    println!("Thanks for running!");
    println!
    ("-----");
}

```

Die EC2 Impl-Struktur dient als Automock-Punkt für Tests, und ihre Funktionen umschließen die EC2 SDK-Aufrufe.

```

use std::{net::Ipv4Addr, time::Duration};

use aws_sdk_ec2::{
    client::Waiters,
    error::ProvideErrorMetadata,
    operation::{
        allocate_address::AllocateAddressOutput,
        associate_address::AssociateAddressOutput,
    }
};

```

```
    },
    types::{
        DomainType, Filter, Image, Instance, InstanceType, IpPermission, IpRange,
        KeyPairInfo,
        SecurityGroup, Tag,
    },
    Client as EC2Client,
};
use aws_sdk_ssm::types::Parameter;
use aws_smithy_runtime_api::client::waiters::error::WaiterError;

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

#[cfg(not(test))]
pub use EC2Impl as EC2;

#[cfg(test)]
pub use MockEC2Impl as EC2;

#[derive(Clone)]
pub struct EC2Impl {
    pub client: EC2Client,
}

#[cfg_attr(test, automock)]
impl EC2Impl {
    pub fn new(client: EC2Client) -> Self {
        EC2Impl { client }
    }

    pub async fn create_key_pair(&self, name: String) -> Result<(KeyPairInfo,
String), EC2Error> {
        tracing::info!("Creating key pair {name}");
        let output = self.client.create_key_pair().key_name(name).send().await?;
        let info = KeyPairInfo::builder()
            .set_key_name(output.key_name)
            .set_key_fingerprint(output.key_fingerprint)
            .set_key_pair_id(output.key_pair_id)
            .build();
        let material = output
            .key_material
            .ok_or_else(|| EC2Error::new("Create Key Pair has no key material"))?;
        Ok((info, material))
    }
}
```

```
}

pub async fn list_key_pair(&self) -> Result<Vec<KeyPairInfo>, EC2Error> {
    let output = self.client.describe_key_pairs().send().await?;
    Ok(output.key_pairs.unwrap_or_default())
}

pub async fn delete_key_pair(&self, key_name: &str) -> Result<(), EC2Error> {
    let key_name: String = key_name.into();
    tracing::info!("Deleting key pair {key_name}");
    self.client
        .delete_key_pair()
        .key_name(key_name)
        .send()
        .await?;
    Ok(())
}

pub async fn create_security_group(
    &self,
    name: &str,
    description: &str,
) -> Result<SecurityGroup, EC2Error> {
    tracing::info!("Creating security group {name}");
    let create_output = self
        .client
        .create_security_group()
        .group_name(name)
        .description(description)
        .send()
        .await
        .map_err(EC2Error::from)?;

    let group_id = create_output
        .group_id
        .ok_or_else(|| EC2Error::new("Missing security group id after
creation"))?;

    let group = self
        .describe_security_group(&group_id)
        .await?
        .ok_or_else(|| {
            EC2Error::new(format!("Could not find security group with id
{group_id}"))
        })
}
```

```

    })?;

    tracing::info!("Created security group {name} as {group_id}");

    Ok(group)
}

/// Find a single security group, by ID. Returns Err if multiple groups are
found.
pub async fn describe_security_group(
    &self,
    group_id: &str,
) -> Result<Option<SecurityGroup>, EC2Error> {
    let group_id: String = group_id.into();
    let describe_output = self
        .client
        .describe_security_groups()
        .group_ids(&group_id)
        .send()
        .await?;

    let mut groups = describe_output.security_groups.unwrap_or_default();

    match groups.len() {
        0 => Ok(None),
        1 => Ok(Some(groups.remove(0))),
        _ => Err(EC2Error::new(format!(
            "Expected single group for {group_id}"
        ))),
    }
}

/// Add an ingress rule to a security group explicitly allowing IPv4 address
/// as {ip}/32 over TCP port 22.
pub async fn authorize_security_group_ssh_ingress(
    &self,
    group_id: &str,
    ingress_ips: Vec<Ipv4Addr>,
) -> Result<(), EC2Error> {
    tracing::info!("Authorizing ingress for security group {group_id}");
    self.client
        .authorize_security_group_ingress()
        .group_id(group_id)
        .set_ip_permissions(Some(

```

```

        ingress_ips
            .into_iter()
            .map(|ip| {
                IpPermission::builder()
                    .ip_protocol("tcp")
                    .from_port(22)
                    .to_port(22)
                    .ip_ranges(IpRange::builder().cidr_ip(format!
({ip}/32)).build())
                    .build()
            })
            .collect(),
    ))
    .send()
    .await?;
    Ok(())
}

pub async fn delete_security_group(&self, group_id: &str) -> Result<(),
EC2Error> {
    tracing::info!("Deleting security group {group_id}");
    self.client
        .delete_security_group()
        .group_id(group_id)
        .send()
        .await?;
    Ok(())
}

pub async fn list_images(&self, ids: Vec<Parameter>) -> Result<Vec<Image>,
EC2Error> {
    let image_ids = ids.into_iter().filter_map(|p| p.value).collect();
    let output = self
        .client
        .describe_images()
        .set_image_ids(Some(image_ids))
        .send()
        .await?;

    let images = output.images.unwrap_or_default();
    if images.is_empty() {
        Err(EC2Error::new("No images for selected AMIs"))
    } else {
        Ok(images)
    }
}

```

```

    }
}

/// List instance types that match an image's architecture and are free tier
eligible.
pub async fn list_instance_types(&self, image: &Image) ->
Result<Vec<InstanceType>, EC2Error> {
    let architecture = format!(
        "{}",
        image.architecture().ok_or_else(|| EC2Error::new(format!(
            "Image {:?} does not have a listed architecture",
            image.image_id()
        )))?
    );
    let free_tier_eligible_filter = Filter::builder()
        .name("free-tier-eligible")
        .values("false")
        .build();
    let supported_architecture_filter = Filter::builder()
        .name("processor-info.supported-architecture")
        .values(architecture)
        .build();
    let response = self
        .client
        .describe_instance_types()
        .filters(free_tier_eligible_filter)
        .filters(supported_architecture_filter)
        .send()
        .await?;

    Ok(response
        .instance_types
        .unwrap_or_default()
        .into_iter()
        .filter_map(|iti| iti.instance_type)
        .collect())
}

pub async fn create_instance<'a>(
    &self,
    image_id: &'a str,
    instance_type: InstanceType,
    key_pair: &'a KeyPairInfo,
    security_groups: Vec<&'a SecurityGroup>,

```

```
) -> Result<String, EC2Error> {
    let run_instances = self
        .client
        .run_instances()
        .image_id(image_id)
        .instance_type(instance_type)
        .key_name(
            key_pair
                .key_name()
                .ok_or_else(|| EC2Error::new("Missing key name when launching
instance"))?,
        )
        .set_security_group_ids(Some(
            security_groups
                .iter()
                .filter_map(|sg| sg.group_id.clone())
                .collect(),
        ))
        .min_count(1)
        .max_count(1)
        .send()
        .await?;

    if run_instances.instances().is_empty() {
        return Err(EC2Error::new("Failed to create instance"));
    }

    let instance_id = run_instances.instances()[0].instance_id().unwrap();
    let response = self
        .client
        .create_tags()
        .resources(instance_id)
        .tags(
            Tag::builder()
                .key("Name")
                .value("From SDK Examples")
                .build(),
        )
        .send()
        .await;

    match response {
        Ok(_) => tracing::info!("Created {instance_id} and applied tags."),
        Err(err) => {
```



```

        tracing::info!("Error applying tags to {instance_id}: {err:?}");
        return Err(err.into());
    }
}

tracing::info!("Instance is created.");

Ok(instance_id.to_string())
}

/// Wait for an instance to be ready and status ok (default wait 60 seconds)
pub async fn wait_for_instance_ready(
    &self,
    instance_id: &str,
    duration: Option<Duration>,
) -> Result<(), EC2Error> {
    self.client
        .wait_until_instance_status_ok()
        .instance_ids(instance_id)
        .wait(duration.unwrap_or(Duration::from_secs(60)))
        .await
        .map_err(|err| match err {
            WaiterError::ExceededMaxWait(exceeded) => EC2Error(format!(
                "Exceeded max time ({}s) waiting for instance to start.",
                exceeded.max_wait().as_secs()
            )),
            _ => EC2Error::from(err),
        })?;
    Ok(())
}

pub async fn describe_instance(&self, instance_id: &str) -> Result<Instance,
EC2Error> {
    let response = self
        .client
        .describe_instances()
        .instance_ids(instance_id)
        .send()
        .await?;

    let instance = response
        .reservations()
        .first()

```

```
        .ok_or_else(|| EC2Error::new(format!("No instance reservations for
{instance_id}")))?
        .instances()
        .first()
        .ok_or_else(|| {
            EC2Error::new(format!("No instances in reservation for
{instance_id}"))
        })?;

    Ok(instance.clone())
}

pub async fn start_instance(&self, instance_id: &str) -> Result<(), EC2Error> {
    tracing::info!("Starting instance {instance_id}");

    self.client
        .start_instances()
        .instance_ids(instance_id)
        .send()
        .await?;

    tracing::info!("Started instance.");

    Ok(())
}

pub async fn stop_instance(&self, instance_id: &str) -> Result<(), EC2Error> {
    tracing::info!("Stopping instance {instance_id}");

    self.client
        .stop_instances()
        .instance_ids(instance_id)
        .send()
        .await?;

    self.wait_for_instance_stopped(instance_id, None).await?;

    tracing::info!("Stopped instance.");

    Ok(())
}

pub async fn reboot_instance(&self, instance_id: &str) -> Result<(), EC2Error> {
```

```
        self.client
            .reboot_instances()
            .instance_ids(instance_id)
            .send()
            .await?;

        Ok(())
    }

    pub async fn wait_for_instance_stopped(
        &self,
        instance_id: &str,
        duration: Option<Duration>,
    ) -> Result<(), EC2Error> {
        self.client
            .wait_until_instance_stopped()
            .instance_ids(instance_id)
            .wait(duration.unwrap_or(Duration::from_secs(60)))
            .await
            .map_err(|err| match err {
                WaiterError::ExceededMaxWait(exceeded) => EC2Error(format!(
                    "Exceeded max time ({}s) waiting for instance to stop.",
                    exceeded.max_wait().as_secs(),
                )),
                _ => EC2Error::from(err),
            })?;
        Ok(())
    }

    pub async fn delete_instance(&self, instance_id: &str) -> Result<(), EC2Error> {
        tracing::info!("Deleting instance with id {instance_id}");
        self.stop_instance(instance_id).await?;
        self.client
            .terminate_instances()
            .instance_ids(instance_id)
            .send()
            .await?;
        self.wait_for_instance_terminated(instance_id).await?;
        tracing::info!("Terminated instance with id {instance_id}");
        Ok(())
    }
}
```

```
    async fn wait_for_instance_terminated(&self, instance_id: &str) -> Result<(),
    EC2Error> {
        self.client
            .wait_until_instance_terminated()
            .instance_ids(instance_id)
            .wait(Duration::from_secs(60))
            .await
            .map_err(|err| match err {
                WaiterError::ExceededMaxWait(exceeded) => EC2Error(format!(
                    "Exceeded max time ({}s) waiting for instance to terminate.",
                    exceeded.max_wait().as_secs(),
                )),
                _ => EC2Error::from(err),
            })?;
        Ok(())
    }

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

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

    pub async fn associate_ip_address(
        &self,
        allocation_id: &str,
        instance_id: &str,
    ) -> Result<AssociateAddressOutput, EC2Error> {
        let response = self
            .client
```

```

        .associate_address()
        .allocation_id(allocation_id)
        .instance_id(instance_id)
        .send()
        .await?;
    Ok(response)
}

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

#[derive(Debug)]
pub struct EC2Error(String);
impl EC2Error {
    pub fn new(value: impl Into<String>) -> Self {
        EC2Error(value.into())
    }

    pub fn add_message(self, message: impl Into<String>) -> Self {
        EC2Error(format!("{}: {}", message.into(), self.0))
    }
}

impl<T: ProvideErrorMetadata> From<T> for EC2Error {
    fn from(value: T) -> Self {
        EC2Error(format!(
            "{}: {}",
            value
                .code()
                .map(String::from)
                .unwrap_or("unknown code".into()),
            value
                .message()
                .map(String::from)
                .unwrap_or("missing reason".into()),
        ))
    }
}

```

```

    }
}

impl std::error::Error for EC2Error {}

impl std::fmt::Display for EC2Error {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(f, "{}", self.0)
    }
}
}

```

Die SSM-Struktur dient als Automock-Punkt für Tests, und ihre Funktionen umschließen SSM-SDK-Aufrufe.

```

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::

```

Das Szenario verwendet mehrere Strukturen im „Manager“-Stil, um den Zugriff auf Ressourcen zu regeln, die im gesamten Szenario erstellt und gelöscht werden.

```

use aws_sdk_ec2::operation::{
    allocate_address::AllocateAddressOutput,
    associate_address::AssociateAddressOutput,
};

use crate::ec2::{EC2Error, EC2};

/// ElasticIpManager tracks the lifecycle of a public IP address, including its
/// allocation from the global pool and association with a specific instance.
#[derive(Debug, Default)]
pub struct ElasticIpManager {
    elastic_ip: Option<AllocateAddressOutput>,
    association: Option<AssociateAddressOutput>,
}

impl ElasticIpManager {
    pub fn has_allocation(&self) -> bool {
        self.elastic_ip.is_some()
    }

    pub fn public_ip(&self) -> &str {
        if let Some(allocation) = &self.elastic_ip {

```

```
        if let Some(addr) = allocation.public_ip() {
            return addr;
        }
    }
    "0.0.0.0"
}

pub async fn allocate(&mut self, ec2: &EC2) -> Result<(), EC2Error> {
    let allocation = ec2.allocate_ip_address().await?;
    self.elastic_ip = Some(allocation);
    Ok(())
}

pub async fn associate(&mut self, ec2: &EC2, instance_id: &str) -> Result<(),
EC2Error> {
    if let Some(allocation) = &self.elastic_ip {
        if let Some(allocation_id) = allocation.allocation_id() {
            let association = ec2.associate_ip_address(allocation_id,
instance_id).await?;
            self.association = Some(association);
            return Ok(());
        }
    }
    Err(EC2Error::new("No ip address allocation to associate"))
}

pub async fn remove(mut self, ec2: &EC2) -> Result<(), EC2Error> {
    if let Some(association) = &self.association {
        if let Some(association_id) = association.association_id() {
            ec2.disassociate_ip_address(association_id).await?;
        }
    }
    self.association = None;
    if let Some(allocation) = &self.elastic_ip {
        if let Some(allocation_id) = allocation.allocation_id() {
            ec2.deallocate_ip_address(allocation_id).await?;
        }
    }
    self.elastic_ip = None;
    Ok(())
}
}
```



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

use aws_sdk_ec2::types::{Instance, InstanceType, KeyPairInfo, SecurityGroup};

use crate::ec2::{EC2Error, EC2};

/// InstanceManager wraps the lifecycle of an EC2 Instance.
#[derive(Debug, Default)]
pub struct InstanceManager {
    instance: Option<Instance>,
}

impl InstanceManager {
    pub fn instance_id(&self) -> &str {
        if let Some(instance) = &self.instance {
            if let Some(id) = instance.instance_id() {
                return id;
            }
        }
        "Unknown"
    }

    pub fn instance_name(&self) -> &str {
        if let Some(instance) = &self.instance {
            if let Some(tag) = instance.tags().iter().find(|e| e.key() ==
Some("Name")) {
                if let Some(value) = tag.value() {
                    return value;
                }
            }
        }
        "Unknown"
    }

    pub fn instance_ip(&self) -> &str {
        if let Some(instance) = &self.instance {
            if let Some(public_ip_address) = instance.public_ip_address() {
                return public_ip_address;
            }
        }
        "0.0.0.0"
    }

    pub fn instance_display_name(&self) -> String {
```

```
        format!("{}", ({}))", self.instance_name(), self.instance_id())
    }

    /// Create an EC2 instance with the given ID on a given type, using a
    /// generated KeyPair and applying a list of security groups.
    pub async fn create(
        &mut self,
        ec2: &EC2,
        image_id: &str,
        instance_type: InstanceType,
        key_pair: &KeyPairInfo,
        security_groups: Vec<&SecurityGroup>,
    ) -> Result<(), EC2Error> {
        let instance_id = ec2
            .create_instance(image_id, instance_type, key_pair, security_groups)
            .await?;
        let instance = ec2.describe_instance(&instance_id).await?;
        self.instance = Some(instance);
        Ok(())
    }

    /// Start the managed EC2 instance, if present.
    pub async fn start(&self, ec2: &EC2) -> Result<(), EC2Error> {
        if self.instance.is_some() {
            ec2.start_instance(self.instance_id()).await?;
        }
        Ok(())
    }

    /// Stop the managed EC2 instance, if present.
    pub async fn stop(&self, ec2: &EC2) -> Result<(), EC2Error> {
        if self.instance.is_some() {
            ec2.stop_instance(self.instance_id()).await?;
        }
        Ok(())
    }

    pub async fn reboot(&self, ec2: &EC2) -> Result<(), EC2Error> {
        if self.instance.is_some() {
            ec2.reboot_instance(self.instance_id()).await?;
            ec2.wait_for_instance_stopped(self.instance_id(), None)
                .await?;
            ec2.wait_for_instance_ready(self.instance_id(), None)
                .await?;
        }
    }
}
```

```

    }
    Ok(())
}

/// Terminate and delete the managed EC2 instance, if present.
pub async fn delete(self, ec2: &EC2) -> Result<(), EC2Error> {
    if self.instance.is_some() {
        ec2.delete_instance(self.instance_id()).await?;
    }
    Ok(())
}
}

impl Display for InstanceManager {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        if let Some(instance) = &self.instance {
            writeln!(f, "\tID: {}", instance.instance_id().unwrap_or("Unknown"))?;
            writeln!(
                f,
                "\tImage ID: {}",
                instance.image_id().unwrap_or("Unknown")
            )?;
            writeln!(
                f,
                "\tInstance type: {}",
                instance
                    .instance_type()
                    .map(|it| format!("{}", it))
                    .unwrap_or("Unknown".to_string())
            )?;
            writeln!(
                f,
                "\tKey name: {}",
                instance.key_name().unwrap_or("Unknown")
            )?;
            writeln!(f, "\tVPC ID: {}", instance.vpc_id().unwrap_or("Unknown"))?;
            writeln!(
                f,
                "\tPublic IP: {}",
                instance.public_ip_address().unwrap_or("Unknown")
            )?;
            let instance_state = instance
                .state
                .as_ref()

```

```

        .map(|is| {
            is.name()
                .map(|isn| format!("{isn}"))
                .unwrap_or("(Unknown)".to_string())
        })
        .unwrap_or("(Unknown)".to_string());
    writeln!(f, "\tState: {instance_state}");
} else {
    writeln!(f, "\tNo loaded instance");
}
Ok(())
}
}

use std::{env, path::PathBuf};

use aws_sdk_ec2::types::KeyPairInfo;

use crate::ec2::{EC2Error, EC2};

use super::util::Util;

/// KeyPairManager tracks a KeyPairInfo and the path the private key has been
/// written to, if it's been created.
#[derive(Debug)]
pub struct KeyPairManager {
    key_pair: KeyPairInfo,
    key_file_path: Option<PathBuf>,
    key_file_dir: PathBuf,
}

impl KeyPairManager {
    pub fn new() -> Self {
        Self::default()
    }

    pub fn key_pair(&self) -> &KeyPairInfo {
        &self.key_pair
    }

    pub fn key_file_path(&self) -> Option<&PathBuf> {
        self.key_file_path.as_ref()
    }
}

```

```

pub fn key_file_dir(&self) -> &PathBuf {
    &self.key_file_dir
}

/// Creates a key pair that can be used to securely connect to an EC2 instance.
/// The returned key pair contains private key information that cannot be
retrieved
/// again. The private key data is stored as a .pem file.
///
/// :param key_name: The name of the key pair to create.
pub async fn create(
    &mut self,
    ec2: &EC2,
    util: &Util,
    key_name: String,
) -> Result<KeyPairInfo, EC2Error> {
    let (key_pair, material) =
ec2.create_key_pair(key_name.clone()).await.map_err(|e| {
        self.key_pair =
KeyPairInfo::builder().key_name(key_name.clone()).build();
        e.add_message(format!("Couldn't create key {key_name}"))
    })?;

    let path = self.key_file_dir.join(format!("{key_name}.pem"));

    // Save the key_pair information immediately, so it can get cleaned up if
write_secure fails.
    self.key_file_path = Some(path.clone());
    self.key_pair = key_pair.clone();

    util.write_secure(&key_name, &path, material)?;

    Ok(key_pair)
}

pub async fn delete(self, ec2: &EC2, util: &Util) -> Result<(), EC2Error> {
    if let Some(key_name) = self.key_pair.key_name() {
        ec2.delete_key_pair(key_name).await?;
        if let Some(key_path) = self.key_file_path() {
            if let Err(err) = util.remove(key_path) {
                eprintln!("Failed to remove {key_path:?} ({err:?})");
            }
        }
    }
}

```

```
    }
    Ok(())
}

pub async fn list(&self, ec2: &EC2) -> Result<Vec<KeyPairInfo>, EC2Error> {
    ec2.list_key_pair().await
}

}

impl Default for KeyPairManager {
    fn default() -> Self {
        KeyPairManager {
            key_pair: KeyPairInfo::builder().build(),
            key_file_path: Default::default(),
            key_file_dir: env::temp_dir(),
        }
    }
}

}

use std::net::Ipv4Addr;

use aws_sdk_ec2::types::SecurityGroup;

use crate::ec2::{EC2Error, EC2};

/// SecurityGroupManager tracks the lifecycle of a SecurityGroup for an instance,
/// including adding a rule to allow SSH from a public IP address.
#[derive(Debug, Default)]
pub struct SecurityGroupManager {
    group_name: String,
    group_description: String,
    security_group: Option<SecurityGroup>,
}

impl SecurityGroupManager {
    pub async fn create(
        &mut self,
        ec2: &EC2,
        group_name: &str,
        group_description: &str,
    ) -> Result<(), EC2Error> {
        self.group_name = group_name.into();
        self.group_description = group_description.into();
    }
}
```

```
        self.security_group = Some(
            ec2.create_security_group(group_name, group_description)
                .await
                .map_err(|e| e.add_message("Couldn't create security group"))?,
        );

        Ok(())
    }

    pub async fn authorize_ingress(&self, ec2: &EC2, ip_address: Ipv4Addr) ->
Result<(), EC2Error> {
        if let Some(sg) = &self.security_group {
            ec2.authorize_security_group_ssh_ingress(
                sg.group_id()
                    .ok_or_else(|| EC2Error::new("Missing security group ID")),
                vec![ip_address],
            )
                .await?;
        };

        Ok(())
    }

    pub async fn delete(self, ec2: &EC2) -> Result<(), EC2Error> {
        if let Some(sg) = &self.security_group {
            ec2.delete_security_group(
                sg.group_id()
                    .ok_or_else(|| EC2Error::new("Missing security group ID")),
            )
                .await?;
        };

        Ok(())
    }

    pub fn group_name(&self) -> &str {
        &self.group_name
    }

    pub fn vpc_id(&self) -> Option<&str> {
        self.security_group.as_ref().and_then(|sg| sg.vpc_id())
    }
}
```

```

    pub fn security_group(&self) -> Option<&SecurityGroup> {
        self.security_group.as_ref()
    }
}

impl std::fmt::Display for SecurityGroupManager {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        match &self.security_group {
            Some(sg) => {
                writeln!(
                    f,
                    "Security group: {}",
                    sg.group_name().unwrap_or("(unknown group)")
                )?;
                writeln!(f, "\tID: {}", sg.group_id().unwrap_or("(unknown group
id)"))?;
                writeln!(f, "\tVPC: {}", sg.vpc_id().unwrap_or("(unknown group
vpc)"))?;
                if !sg.ip_permissions().is_empty() {
                    writeln!(f, "\tInbound Permissions:")?;
                    for permission in sg.ip_permissions() {
                        writeln!(f, "\t\t{permission:?}")?;
                    }
                }
                Ok(())
            }
            None => writeln!(f, "No security group loaded."),
        }
    }
}
}

```

Der Haupteinstiegspunkt für das Szenario.

```

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;
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der API-Referenz zu AWS - SDK für Rust.
 - [AllocateAddress](#)
 - [AssociateAddress](#)
 - [AuthorizeSecurityGroupIngress](#)
 - [CreateKeyPair](#)
 - [CreateSecurityGroup](#)
 - [DeleteKeyPair](#)
 - [DeleteSecurityGroup](#)
 - [DescribeImages](#)
 - [DescribeInstanceTypes](#)
 - [DescribeInstances](#)
 - [DescribeKeyPairs](#)
 - [DescribeSecurityGroups](#)
 - [DisassociateAddress](#)
 - [ReleaseAddress](#)
 - [RunInstances](#)
 - [StartInstances](#)
 - [StopInstances](#)
 - [TerminateInstances](#)
 - [UnmonitorInstances](#)

Aktionen

AllocateAddress

Das folgende Codebeispiel zeigt die Verwendung `AllocateAddress`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

- Einzelheiten zur API finden Sie [AllocateAddress](#) in der API-Referenz zum AWS SDK für Rust.

AssociateAddress

Das folgende Codebeispiel zeigt, wie man es benutzt `AssociateAddress`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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)
}

```

- Einzelheiten zur API finden Sie [AssociateAddress](#) in der API-Referenz zum AWS SDK für Rust.

AuthorizeSecurityGroupIngress

Das folgende Codebeispiel zeigt, wie man es benutzt `AuthorizeSecurityGroupIngress`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

/// 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(())
}

```

- Einzelheiten zur API finden Sie [AuthorizeSecurityGroupIngress](#) in der API-Referenz zum AWS SDK für Rust.

CreateKeyPair

Das folgende Codebeispiel zeigt, wie man es benutzt `CreateKeyPair`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Rust-Implementierung, die `create_key_pair` des EC2 Clients aufruft und das zurückgegebene Material extrahiert.

```

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))
}

```

Eine Funktion, die das `create_key`-Impl aufruft und den privaten PEM-Schlüssel sicher speichert.

```

/// Creates a key pair that can be used to securely connect to an EC2 instance.
/// The returned key pair contains private key information that cannot be
retrieved
/// again. The private key data is stored as a .pem file.
///
/// :param key_name: The name of the key pair to create.
pub async fn create(
    &mut self,
    ec2: &EC2,
    util: &Util,
    key_name: String,
) -> Result<KeyPairInfo, EC2Error> {
    let (key_pair, material) =
ec2.create_key_pair(key_name.clone()).await.map_err(|e| {
        self.key_pair =
KeyPairInfo::builder().key_name(key_name.clone()).build();
        e.add_message(format!("Couldn't create key {key_name}"))
    })?;

    let path = self.key_file_dir.join(format!("{key_name}.pem"));

    // Save the key_pair information immediately, so it can get cleaned up if
write_secure fails.
    self.key_file_path = Some(path.clone());
    self.key_pair = key_pair.clone();

    util.write_secure(&key_name, &path, material)?;

    Ok(key_pair)
}

```

- Einzelheiten zur API finden Sie [CreateKeyPair](#) in der API-Referenz zum AWS SDK für Rust.

CreateSecurityGroup

Das folgende Codebeispiel zeigt, wie man es benutzt `CreateSecurityGroup`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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)
    }

```

- Einzelheiten zur API finden Sie [CreateSecurityGroup](#) in der API-Referenz zum AWS SDK für Rust.

CreateTags

Das folgende Codebeispiel zeigt, wie man es benutzt `CreateTags`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

In diesem Beispiel wird das Name-Tag nach dem Erstellen einer Instanz angewendet.

```

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())
}
```

- Einzelheiten zur API finden Sie [CreateTags](#) in der API-Referenz zum AWS SDK für Rust.

DeleteKeyPair

Das folgende Codebeispiel zeigt, wie man es benutzt `DeleteKeyPair`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Wrapper für `delete_key`, der auch den unterstützenden privaten PEM-Schlüssel entfernt.

```
pub async fn delete(self, ec2: &EC2, util: &Util) -> Result<(), EC2Error> {
    if let Some(key_name) = self.key_pair.key_name() {
        ec2.delete_key_pair(key_name).await?;
        if let Some(key_path) = self.key_file_path() {
            if let Err(err) = util.remove(key_path) {
                eprintln!("Failed to remove {key_path:?} ({err:?})");
            }
        }
    }
    Ok(())
}
```

```
pub async fn delete_key_pair(&self, key_name: &str) -> Result<(), EC2Error> {
    let key_name: String = key_name.into();
    tracing::info!("Deleting key pair {key_name}");
    self.client
        .delete_key_pair()
        .key_name(key_name)
        .send()
        .await?;
    Ok(())
}
```

- Einzelheiten zur API finden Sie [DeleteKeyPair](#) in der API-Referenz zum AWS SDK für Rust.

DeleteSecurityGroup

Das folgende Codebeispiel zeigt, wie man es benutzt `DeleteSecurityGroup`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [DeleteSecurityGroup](#) in der API-Referenz zum AWS SDK für Rust.

DeleteSnapshot

Das folgende Codebeispiel zeigt, wie man es benutzt `DeleteSnapshot`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
async fn delete_snapshot(client: &Client, id: &str) -> Result<(), Error> {
```

```
client.delete_snapshot().snapshot_id(id).send().await?;  
  
println!("Deleted");  
  
Ok(())  
}
```

- Einzelheiten zur API finden Sie [DeleteSnapshot](#) in der API-Referenz zum AWS SDK für Rust.

DescribeImages

Das folgende Codebeispiel zeigt, wie man es benutzt `DescribeImages`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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)  
    }  
}
```

Verwenden Sie die Funktion `list_images` mit SSM, um basierend auf Ihrer Umgebung zu begrenzen. Weitere Informationen zu SSM finden Sie unter `_ssm_` [_section.html](https://docs.aws.amazon.com/systems-manager/latest/userguide/example-GetParameters). <https://docs.aws.amazon.com/systems-manager/latest/userguide/example-GetParameters>

```

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)
}

```

- Einzelheiten zur API finden Sie [DescribeImages](#) in der API-Referenz zum AWS SDK für Rust.

DescribeInstanceStatus

Das folgende Codebeispiel zeigt, wie man es benutzt `DescribeInstanceStatus`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

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(())
}

```

- Einzelheiten zur API finden Sie [DescribeInstanceStatus](#) in der API-Referenz zum AWS SDK für Rust.

DescribeInstanceTypes

Das folgende Codebeispiel zeigt, wie man es benutzt `DescribeInstanceTypes`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

- Einzelheiten zur API finden Sie [DescribeInstanceTypes](#) in der API-Referenz zum AWS SDK für Rust.

DescribeInstances

Das folgende Codebeispiel zeigt, wie man es benutzt `DescribeInstances`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Rufen Sie Details für eine EC2 Instanz ab.

```
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())
}
```

Rufen Sie nach dem Erstellen einer EC2 Instanz deren Details ab und speichern Sie sie.

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

- Einzelheiten zur API finden Sie [DescribeInstances](#) in der API-Referenz zum AWS SDK für Rust.

DescribeKeyPairs

Das folgende Codebeispiel zeigt, wie man es benutzt `DescribeKeyPairs`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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())
}
```


- Einzelheiten zur API finden Sie [DescribeKeyPairs](#) in der API-Referenz zum AWS SDK für Rust.

DescribeRegions

Das folgende Codebeispiel zeigt, wie man es benutzt `DescribeRegions`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [DescribeRegions](#) in der API-Referenz zum AWS SDK für Rust.

DescribeSecurityGroups

Das folgende Codebeispiel zeigt, wie man es benutzt `DescribeSecurityGroups`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

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");
        }
    }
}


```

- Einzelheiten zur API finden Sie [DescribeSecurityGroups](#) in der API-Referenz zum AWS SDK für Rust.

DescribeSnapshots

Das folgende Codebeispiel zeigt, wie man es benutztDescribeSnapshots.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Zeigt den Status eines Snapshots an.

```
async fn show_state(client: &Client, id: &str) -> Result<(), Error> {
    let resp = client
        .describe_snapshots()
        .filters(Filter::builder().name("snapshot-id").values(id).build())
        .send()
        .await?;

    println!(
        "State: {}",
        resp.snapshots().first().unwrap().state().unwrap().as_ref()
    );

    Ok(())
}
```

```
async fn show_snapshots(client: &Client) -> Result<(), Error> {
    // "self" represents your account ID.
    // You can list the snapshots for any account by replacing
    // "self" with that account ID.
    let resp = client.describe_snapshots().owner_ids("self").send().await?;
    let snapshots = resp.snapshots();
    let length = snapshots.len();

    for snapshot in snapshots {
        println!(
            "ID:          {}",
            snapshot.snapshot_id().unwrap_or_default()
        );
        println!(
            "Description: {}",
            snapshot.description().unwrap_or_default()
        );
    }
}
```

```

        );
        println!("State:      {}", snapshot.state().unwrap().as_ref());
        println!();
    }

    println!();
    println!("Found {} snapshot(s)", length);
    println!();

    Ok(())
}

```

- Einzelheiten zur API finden Sie [DescribeSnapshots](#) in der API-Referenz zum AWS SDK für Rust.

DisassociateAddress

Das folgende Codebeispiel zeigt, wie man es benutzt `DisassociateAddress`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

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

```

- Einzelheiten zur API finden Sie [DisassociateAddress](#) in der API-Referenz zum AWS SDK für Rust.

RebootInstances

Das folgende Codebeispiel zeigt, wie man es benutzt `RebootInstances`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
pub async fn reboot(&self, ec2: &EC2) -> Result<(), EC2Error> {
    if self.instance.is_some() {
        ec2.reboot_instance(self.instance_id()).await?;
        ec2.wait_for_instance_stopped(self.instance_id(), None)
            .await?;
        ec2.wait_for_instance_ready(self.instance_id(), None)
            .await?;
    }
    Ok(())
}
```

```
pub async fn reboot_instance(&self, instance_id: &str) -> Result<(), EC2Error> {
    tracing::info!("Rebooting instance {instance_id}");

    self.client
        .reboot_instances()
        .instance_ids(instance_id)
        .send()
        .await?;

    Ok(())
}
```

Kellner müssen sich beispielsweise mithilfe der Waiters API im Status „Angehalten“ und „Bereit“ befinden. Die Verwendung der Waiters API erfordert `use aws_sdk_ec2: :client: :Waiters`` in der Rust-Datei.

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

- Einzelheiten zur API finden Sie in der API-Referenz zum SDK für Rust. [RebootInstancesAWS](#)

ReleaseAddress

Das folgende Codebeispiel zeigt, wie man es benutzt `ReleaseAddress`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

- Einzelheiten zur API finden Sie [ReleaseAddress](#) in der API-Referenz zum AWS SDK für Rust.

RunInstances

Das folgende Codebeispiel zeigt, wie man es benutzt `RunInstances`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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())
}
```

- Einzelheiten zur API finden Sie [RunInstances](#) in der API-Referenz zum AWS SDK für Rust.

StartInstances

Das folgende Codebeispiel zeigt, wie man es benutzt `StartInstances`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Starten Sie eine EC2 Instance anhand der Instanz-ID.

```
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(())
}
```

Warten Sie mithilfe der Waiters API, bis sich eine Instance im Status „Bereit“ und „OK“ befindet. Die Verwendung der Waiters API erfordert `use aws_sdk_ec2: :client: :Waiters` in der Rust-Datei.

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

- Einzelheiten zur API finden Sie in der API-Referenz zum SDK für Rust. [StartInstancesAWS](#)

StopInstances

Das folgende Codebeispiel zeigt, wie man es benutzt StopInstances.

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

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(())
}

```

Warten Sie mithilfe der Waiters API, bis sich eine Instance im Status „Gestoppt“ befindet. Die Verwendung der Waiters API erfordert `use aws_sdk_ec2: :client: :Waiters` in der Rust-Datei.

```

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(())
}

```

- Einzelheiten zur API finden Sie in der API-Referenz zum SDK für Rust. [StopInstancesAWS](#)

TerminateInstances

Das folgende Codebeispiel zeigt, wie man es benutzt TerminateInstances.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

Warten Sie mithilfe der Waiters API, bis sich eine Instance im Status „Beendet“ befindet. Die Verwendung der Waiters API erfordert `use aws_sdk_ec2: :client: :Waiters` in der Rust-Datei.

```
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(())
}
```

- Einzelheiten zur API finden Sie in der API-Referenz zum SDK für Rust.
[TerminateInstancesAWS](#)

Amazon ECR-Beispiele mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe des AWS SDK für Rust mit Amazon ECR Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

DescribeRepositories

Das folgende Codebeispiel zeigt die Verwendung `DescribeRepositories`.

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [DescribeRepositories](#) in der API-Referenz zum AWS SDK für Rust.

ListImages

Das folgende Codebeispiel zeigt, wie man es benutzt `ListImages`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [ListImages](#) in der API-Referenz zum AWS SDK für Rust.

Amazon ECS-Beispiele mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe des AWS SDK für Rust mit Amazon ECS Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Themen


- [Aktionen](#)

Aktionen

CreateCluster

Das folgende Codebeispiel zeigt die Verwendung `CreateCluster`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [CreateCluster](#) in der API-Referenz zum AWS SDK für Rust.

DeleteCluster

Das folgende Codebeispiel zeigt, wie man es benutzt `DeleteCluster`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```



```
}
```

- Einzelheiten zur API finden Sie [DeleteCluster](#) in der API-Referenz zum AWS SDK für Rust.

DescribeClusters

Das folgende Codebeispiel zeigt, wie man es benutzt `DescribeClusters`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [DescribeClusters](#) in der API-Referenz zum AWS SDK für Rust.

Amazon EKS-Beispiele mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe des AWS SDK für Rust mit Amazon EKS Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

CreateCluster

Das folgende Codebeispiel zeigt die Verwendung `CreateCluster`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}

```

- Einzelheiten zur API finden Sie [CreateCluster](#) in der API-Referenz zum AWS SDK für Rust.

DeleteCluster

Das folgende Codebeispiel zeigt, wie man es benutzt `DeleteCluster`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

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(())
}

```

- Einzelheiten zur API finden Sie [DeleteCluster](#) in der API-Referenz zum AWS SDK für Rust.

AWS Glue Beispiele für die Verwendung von SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie das AWS SDK für Rust mit verwenden AWS Glue.

Bei Grundlagen handelt es sich um Code-Beispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo AWS Glue

Die folgenden Codebeispiele veranschaulichen, wie Sie mit der Verwendung von AWS Glue beginnen.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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)),
    }
}
```

- Einzelheiten zur API finden Sie [ListJobs](#) in der API-Referenz zum AWS SDK für Rust.

Themen

- [Grundlagen](#)
- [Aktionen](#)

Grundlagen

Erlernen der Grundlagen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie einen Crawler, der einen öffentlichen Amazon-S3-Bucket crawlt und eine Datenbank mit CSV-formatierten Metadaten generiert.
- Listen Sie Informationen zu Datenbanken und Tabellen in Ihrem auf AWS Glue Data Catalog.
- Erstellen Sie einen Auftrag, um CSV-Daten aus dem S3-Bucket zu extrahieren, die Daten umzuwandeln und die JSON-formatierte Ausgabe in einen anderen S3-Bucket zu laden.
- Listen Sie Informationen zu Auftragsausführungen auf, zeigen Sie transformierte Daten an und bereinigen Sie Ressourcen.

Weitere Informationen finden Sie unter [Tutorial: Erste Schritte mit AWS Glue Studio](#).

SDK für Rust

Note

Es gibt mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen Sie einen Crawler und führen Sie ihn aus, der einen öffentlichen Amazon Simple Storage Service (Amazon S3)-Bucket crawlt und eine Metadatendatenbank generiert, die die gefundenen CSV-formatierten Daten beschreibt.

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

```

Listen Sie Informationen zu Datenbanken und Tabellen in Ihrem auf 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();

```

Erstellen und führen Sie einen Auftrag aus, der CSV-Daten aus dem Amazon-S3-Quell-Bucket extrahiert, sie durch Entfernen und Umbenennen von Feldern transformiert und die JSON-formatierte Ausgabe in einen anderen Amazon-S3-Bucket lädt.

```

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();

```

Löscht alle Ressourcen, die von der Demo erstellt wurden.

```

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)?;
```


- Weitere API-Informationen finden Sie in den folgenden Themen der API-Referenz zu AWS - SDK für Rust.
 - [CreateCrawler](#)
 - [CreateJob](#)
 - [DeleteCrawler](#)
 - [DeleteDatabase](#)
 - [DeleteJob](#)
 - [DeleteTable](#)
 - [GetCrawler](#)
 - [GetDatabase](#)
 - [GetDatabases](#)
 - [GetJob](#)
 - [GetJobRun](#)
 - [GetJobRuns](#)
 - [GetTables](#)
 - [ListJobs](#)
 - [StartCrawler](#)
 - [StartJobRun](#)

Aktionen

CreateCrawler

Das folgende Codebeispiel zeigt die Verwendung `CreateCrawler`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

- Einzelheiten zur API finden Sie [CreateCrawler](#) in der API-Referenz zum AWS SDK für Rust.

CreateJob

Das folgende Codebeispiel zeigt, wie man es benutzt `CreateJob`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

- Einzelheiten zur API finden Sie [CreateJob](#) in der API-Referenz zum AWS SDK für Rust.

DeleteCrawler

Das folgende Codebeispiel zeigt, wie man es benutzt `DeleteCrawler`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.


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

- Einzelheiten zur API finden Sie [DeleteCrawler](#) in der API-Referenz zum AWS SDK für Rust.

DeleteDatabase

Das folgende Codebeispiel zeigt, wie man es benutzt `DeleteDatabase`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

- Einzelheiten zur API finden Sie [DeleteDatabase](#) in der API-Referenz zum AWS SDK für Rust.

DeleteJob

Das folgende Codebeispiel zeigt, wie man es benutzt `DeleteJob`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

- Einzelheiten zur API finden Sie [DeleteJob](#) in der API-Referenz zum AWS SDK für Rust.

DeleteTable

Das folgende Codebeispiel zeigt, wie man es benutzt `DeleteTable`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

```
}
```

- Einzelheiten zur API finden Sie [DeleteTable](#) in der API-Referenz zum AWS SDK für Rust.

GetCrawler

Das folgende Codebeispiel zeigt, wie man es benutzt `GetCrawler`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

- Einzelheiten zur API finden Sie [GetCrawler](#) in der API-Referenz zum AWS SDK für Rust.

GetDatabase

Das folgende Codebeispiel zeigt, wie man es benutzt `GetDatabase`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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()))?;

```

- Einzelheiten zur API finden Sie [GetDatabase](#) in der API-Referenz zum AWS SDK für Rust.

GetJobRun

Das folgende Codebeispiel zeigt, wie man es benutzt `GetJobRun`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

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();
    }

```

- Einzelheiten zur API finden Sie [GetJobRun](#) in der API-Referenz zum AWS SDK für Rust.

GetTables

Das folgende Codebeispiel zeigt, wie man es benutzt `GetTables`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

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

let tables = tables.table_list();

```

- Einzelheiten zur API finden Sie [GetTables](#) in der API-Referenz zum AWS SDK für Rust.

ListJobs

Das folgende Codebeispiel zeigt, wie man es benutzt `ListJobs`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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)),
    }
}
```

- Einzelheiten zur API finden Sie [ListJobs](#) in der API-Referenz zum AWS SDK für Rust.

StartCrawler

Das folgende Codebeispiel zeigt, wie man es benutzt `StartCrawler`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

```

- Einzelheiten zur API finden Sie [StartCrawler](#) in der API-Referenz zum AWS SDK für Rust.

StartJobRun

Das folgende Codebeispiel zeigt, wie man es benutzt `StartJobRun`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

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();
```

- Einzelheiten zur API finden Sie [StartJobRun](#) in der API-Referenz zum AWS SDK für Rust.

IAM-Beispiele mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe des AWS SDK für Rust mit IAM Aktionen ausführen und gängige Szenarien implementieren.

Bei Grundlagen handelt es sich um Code-Beispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo IAM

Die folgenden Codebeispiele veranschaulichen, wie Sie mit der Verwendung von IAM beginnen.

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Von src/bin/hello R.S.

```

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(())
}

```

Von src/ .rsiam-service-lib.

```

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);
    })
    .collect();

Ok(policy_names)
}
```

- Einzelheiten zur API finden Sie [ListPolicies](#) in der API-Referenz zum AWS SDK für Rust.

Themen

- [Grundlagen](#)
- [Aktionen](#)

Grundlagen

Erlernen der Grundlagen

Das folgende Codebeispiel veranschaulicht, wie Sie einen Benutzer erstellen und eine Rolle annehmen lassen.

Warning

Um Sicherheitsrisiken zu vermeiden, sollten Sie IAM-Benutzer nicht zur Authentifizierung verwenden, wenn Sie speziell entwickelte Software entwickeln oder mit echten Daten arbeiten. Verwenden Sie stattdessen den Verbund mit einem Identitätsanbieter wie [AWS IAM Identity Center](#).

- Erstellen Sie einen Benutzer ohne Berechtigungen.
- Erstellen einer Rolle, die die Berechtigung zum Auflisten von Amazon-S3-Buckets für das Konto erteilt.

- Hinzufügen einer Richtlinie, damit der Benutzer die Rolle übernehmen kann.
- Übernehmen Sie die Rolle und listen Sie S3-Buckets mit temporären Anmeldeinformationen auf, und bereinigen Sie dann die Ressourcen.

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
use aws_config::meta::region::RegionProviderChain;
use aws_sdk_iam::Error as iamError;
use aws_sdk_iam::{config::Credentials as iamCredentials, config::Region, Client as iamClient};
use aws_sdk_s3::Client as s3Client;
use aws_sdk_sts::Client as stsClient;
use tokio::time::{sleep, Duration};
use uuid::Uuid;

#[tokio::main]
async fn main() -> Result<(), iamError> {
    let (client, uuid, list_all_buckets_policy_document, inline_policy_document) =
        initialize_variables().await;

    if let Err(e) = run_iam_operations(
        client,
        uuid,
        list_all_buckets_policy_document,
        inline_policy_document,
    )
    .await
    {
        println!("{:?}", e);
    };

    Ok(())
}
```

```

async fn initialize_variables() -> (iamClient, String, String, String) {
    let region_provider = RegionProviderChain::first_try(Region::new("us-west-2"));

    let shared_config = aws_config::from_env().region(region_provider).load().await;
    let client = iamClient::new(&shared_config);
    let uuid = Uuid::new_v4().to_string();

    let list_all_buckets_policy_document = "{
        \"Version\": \"2012-10-17\",
        \"Statement\": [{
            \"Effect\": \"Allow\",
            \"Action\": \"s3:ListAllMyBuckets\",
            \"Resource\": \"arn:aws:s3::*\"}]
    }"
    .to_string();
    let inline_policy_document = "{
        \"Version\": \"2012-10-17\",
        \"Statement\": [{
            \"Effect\": \"Allow\",
            \"Action\": \"sts:AssumeRole\",
            \"Resource\": \"{}\"}]
    }"
    .to_string();

    (
        client,
        uuid,
        list_all_buckets_policy_document,
        inline_policy_document,
    )
}

async fn run_iam_operations(
    client: iamClient,
    uuid: String,
    list_all_buckets_policy_document: String,
    inline_policy_document: String,
) -> Result<(), iamError> {
    let user = iam_service::create_user(&client, &format!("{}", "iam_demo_user_",
    uuid)).await?;
    println!("Created the user with the name: {}", user.user_name());
    let key = iam_service::create_access_key(&client, user.user_name()).await?;

    let assume_role_policy_document = "{

```

```

        \ "Version\ ": \ "2012-10-17\ ",
          \ "Statement\ ": [{
            \ "Effect\ ": \ "Allow\ ",
            \ "Principal\ ": { \ "AWS\ ": \ "{}\ " },
            \ "Action\ ": \ "sts:AssumeRole\ "
          }]
      }"
    .to_string()
    .replace("{} ", user.arn());

let assume_role_role = iam_service::create_role(
    &client,
    &format!("{}", "iam_demo_role_", uuid),
    &assume_role_policy_document,
)
.await?;
println!("Created the role with the ARN: {}", assume_role_role.arn());

let list_all_buckets_policy = iam_service::create_policy(
    &client,
    &format!("{}", "iam_demo_policy_", uuid),
    &list_all_buckets_policy_document,
)
.await?;
println!(
    "Created policy: {}",
    list_all_buckets_policy.policy_name.as_ref().unwrap()
);

let attach_role_policy_result =
    iam_service::attach_role_policy(&client, &assume_role_role,
&list_all_buckets_policy)
    .await?;
println!(
    "Attached the policy to the role: {:?}",
    attach_role_policy_result
);

let inline_policy_name = format!("{}", "iam_demo_inline_policy_", uuid);
let inline_policy_document = inline_policy_document.replace("{} ",
assume_role_role.arn());
iam_service::create_user_policy(&client, &user, &inline_policy_name,
&inline_policy_document)
    .await?;

```



```
println!("Created inline policy.");

//First, fail to list the buckets with the user.
let creds = iamCredentials::from_keys(key.access_key_id(),
key.secret_access_key(), None);
let fail_config = aws_config::from_env()
    .credentials_provider(creds.clone())
    .load()
    .await;
println!("Fail config: {:?}", fail_config);
let fail_client: s3Client = s3Client::new(&fail_config);
match fail_client.list_buckets().send().await {
    Ok(e) => {
        println!("This should not run. {:?}", e);
    }
    Err(e) => {
        println!("Successfully failed with error: {:?}", e)
    }
}

let sts_config = aws_config::from_env()
    .credentials_provider(creds.clone())
    .load()
    .await;
let sts_client: stsClient = stsClient::new(&sts_config);
sleep(Duration::from_secs(10)).await;
let assumed_role = sts_client
    .assume_role()
    .role_arn(assume_role_role.arn())
    .role_session_name(format!("iam_demo_assumerole_session_{uuid}"))
    .send()
    .await;
println!("Assumed role: {:?}", assumed_role);
sleep(Duration::from_secs(10)).await;

let assumed_credentials = iamCredentials::from_keys(
    assumed_role
        .as_ref()
        .unwrap()
        .credentials
        .as_ref()
        .unwrap()
        .access_key_id(),
    assumed_role
```

```

        .as_ref()
        .unwrap()
        .credentials
        .as_ref()
        .unwrap()
        .secret_access_key(),
    Some(
        assumed_role
            .as_ref()
            .unwrap()
            .credentials
            .as_ref()
            .unwrap()
            .session_token
            .clone(),
    ),
);

let succeed_config = aws_config::from_env()
    .credentials_provider(assumed_credentials)
    .load()
    .await;
println!("succeed config: {:?}", succeed_config);
let succeed_client: s3Client = s3Client::new(&succeed_config);
sleep(Duration::from_secs(10)).await;
match succeed_client.list_buckets().send().await {
    Ok(_) => {
        println!("This should now run successfully.")
    }
    Err(e) => {
        println!("This should not run. {:?}", e);
        panic!()
    }
}

//Clean up.
iam_service::detach_role_policy(
    &client,
    assume_role_role.role_name(),
    list_all_buckets_policy.arn().unwrap_or_default(),
)
.await?;
iam_service::delete_policy(&client, list_all_buckets_policy).await?;
iam_service::delete_role(&client, &assume_role_role).await?;

```

```
println!("Deleted role {}", assume_role_role.role_name());
iam_service::delete_access_key(&client, &user, &key).await?;
println!("Deleted key for {}", key.user_name());
iam_service::delete_user_policy(&client, &user, &inline_policy_name).await?;
println!("Deleted inline user policy: {}", inline_policy_name);
iam_service::delete_user(&client, &user).await?;
println!("Deleted user {}", user.user_name());

Ok(())
}
```


- Weitere API-Informationen finden Sie in den folgenden Themen der API-Referenz zu AWS - SDK für Rust.
 - [AttachRolePolicy](#)
 - [CreateAccessKey](#)
 - [CreatePolicy](#)
 - [CreateRole](#)
 - [CreateUser](#)
 - [DeleteAccessKey](#)
 - [DeletePolicy](#)
 - [DeleteRole](#)
 - [DeleteUser](#)
 - [DeleteUserPolicy](#)
 - [DetachRolePolicy](#)
 - [PutUserPolicy](#)

Aktionen

AttachRolePolicy

Das folgende Codebeispiel zeigt, wie man es benutzt `AttachRolePolicy`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.


```
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
}
```

- Einzelheiten zur API finden Sie [AttachRolePolicy](#) in der API-Referenz zum AWS SDK für Rust.

AttachUserPolicy

Das folgende Codebeispiel zeigt, wie man es benutzt `AttachUserPolicy`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}

```

- Einzelheiten zur API finden Sie [AttachUserPolicy](#) in der API-Referenz zum AWS SDK für Rust.

CreateAccessKey

Das folgende Codebeispiel zeigt, wie man es benutzt `CreateAccessKey`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

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())
}

```

- Einzelheiten zur API finden Sie [CreateAccessKey](#) in der API-Referenz zum AWS SDK für Rust.

CreatePolicy

Das folgende Codebeispiel zeigt, wie man es benutzt `CreatePolicy`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

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())
}

```

- Einzelheiten zur API finden Sie [CreatePolicy](#) in der API-Referenz zum AWS SDK für Rust.

CreateRole

Das folgende Codebeispiel zeigt, wie man es benutzt `CreateRole`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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())
}
```

- Einzelheiten zur API finden Sie [CreateRole](#) in der API-Referenz zum AWS SDK für Rust.

CreateServiceLinkedRole

Das folgende Codebeispiel zeigt, wie man es benutzt `CreateServiceLinkedRole`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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)
}
```

- Einzelheiten zur API finden Sie [CreateServiceLinkedRole](#) in der API-Referenz zum AWS SDK für Rust.

CreateUser

Das folgende Codebeispiel zeigt, wie man es benutzt CreateUser.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.


```
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())
}
```

- Einzelheiten zur API finden Sie [CreateUser](#) in der API-Referenz zum AWS SDK für Rust.

DeleteAccessKey

Das folgende Codebeispiel zeigt, wie man es benutzt `DeleteAccessKey`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}

```

- Einzelheiten zur API finden Sie [DeleteAccessKey](#) in der API-Referenz zum AWS SDK für Rust.

DeletePolicy

Das folgende Codebeispiel zeigt, wie man es benutzt `DeletePolicy`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

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


```

- Einzelheiten zur API finden Sie [DeletePolicy](#) in der API-Referenz zum AWS SDK für Rust.

DeleteRole

Das folgende Codebeispiel zeigt, wie man es benutzt `DeleteRole`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.


```
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(())
}
```

- Einzelheiten zur API finden Sie [DeleteRole](#) in der API-Referenz zum AWS SDK für Rust.

DeleteServiceLinkedRole

Das folgende Codebeispiel zeigt, wie man es benutzt `DeleteServiceLinkedRole`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}

```

- Einzelheiten zur API finden Sie [DeleteServiceLinkedRole](#) in der API-Referenz zum AWS SDK für Rust.

DeleteUser

Das folgende Codebeispiel zeigt, wie man es benutzt `DeleteUser`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

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
}

```

- Einzelheiten zur API finden Sie [DeleteUser](#) in der API-Referenz zum AWS SDK für Rust.

DeleteUserPolicy

Das folgende Codebeispiel zeigt, wie man es benutzt `DeleteUserPolicy`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

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(())  
}
```

- Einzelheiten zur API finden Sie [DeleteUserPolicy](#) in der API-Referenz zum AWS SDK für Rust.

DetachRolePolicy

Das folgende Codebeispiel zeigt, wie man es benutzt `DetachRolePolicy`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.


```
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(())  
}
```

- Einzelheiten zur API finden Sie [DetachRolePolicy](#) in der API-Referenz zum AWS SDK für Rust.

DetachUserPolicy

Das folgende Codebeispiel zeigt, wie man es benutzt `DetachUserPolicy`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [DetachUserPolicy](#) in der API-Referenz zum AWS SDK für Rust.

GetAccountPasswordPolicy

Das folgende Codebeispiel zeigt, wie man es benutzt `GetAccountPasswordPolicy`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
pub async fn get_account_password_policy(
```

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

    Ok(response)
}
```

- Einzelheiten zur API finden Sie [GetAccountPasswordPolicy](#) in der API-Referenz zum AWS SDK für Rust.

GetRole

Das folgende Codebeispiel zeigt, wie man es benutzt `GetRole`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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)
}
```

- Einzelheiten zur API finden Sie [GetRole](#) in der API-Referenz zum AWS SDK für Rust.

ListAttachedRolePolicies

Das folgende Codebeispiel zeigt, wie man es benutzt `ListAttachedRolePolicies`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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)
}
```

- Einzelheiten zur API finden Sie [ListAttachedRolePolicies](#) in der API-Referenz zum AWS SDK für Rust.

ListGroups

Das folgende Codebeispiel zeigt, wie man es benutzt `ListGroups`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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)
}
```

- Einzelheiten zur API finden Sie [ListGroups](#) in der API-Referenz zum AWS SDK für Rust.

ListPolicies

Das folgende Codebeispiel zeigt, wie man es benutzt `ListPolicies`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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)
}
```

- Einzelheiten zur API finden Sie [ListPolicies](#) in der API-Referenz zum AWS SDK für Rust.

ListRolePolicies

Das folgende Codebeispiel zeigt, wie man es benutzt `ListRolePolicies`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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)
}
```

- Einzelheiten zur API finden Sie [ListRolePolicies](#) in der API-Referenz zum AWS SDK für Rust.

ListRoles

Das folgende Codebeispiel zeigt, wie man es benutzt `ListRoles`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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)
}
```

- Einzelheiten zur API finden Sie [ListRoles](#) in der API-Referenz zum AWS SDK für Rust.

ListSAMLProviders

Das folgende Codebeispiel zeigt, wie man es benutzt `ListSAMLProviders`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
pub async fn list_saml_providers(
    client: &Client,
) -> Result<ListSamlProvidersOutput, SdkError<ListSAMLProvidersError>> {
    let response = client.list_saml_providers().send().await?;

    Ok(response)
}
```

- Einzelheiten zur API finden Sie unter [Liste SAMLProviders](#) im AWS SDK für die Rust-API-Referenz.

ListUsers

Das folgende Codebeispiel zeigt, wie man es benutzt `ListUsers`.

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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)
}
```

- Einzelheiten zur API finden Sie [ListUsers](#) in der API-Referenz zum AWS SDK für Rust.

AWS IoT Beispiele für die Verwendung von SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie das AWS SDK für Rust mit verwenden AWS IoT.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

DescribeEndpoint

Das folgende Codebeispiel zeigt die Verwendung `DescribeEndpoint`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [DescribeEndpoint](#) in der API-Referenz zum AWS SDK für Rust.

ListThings

Das folgende Codebeispiel zeigt, wie man es benutzt `ListThings`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [ListThings](#) in der API-Referenz zum AWS SDK für Rust.

Kinesis-Beispiele mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe des AWS SDK für Rust mit Kinesis Aktionen ausführen und gängige Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)
- [Serverless-Beispiele](#)

Aktionen

CreateStream

Das folgende Codebeispiel zeigt die Verwendung `CreateStream`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [CreateStream](#) in der API-Referenz zum AWS SDK für Rust.

DeleteStream

Das folgende Codebeispiel zeigt, wie man es benutzt `DeleteStream`.

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
async fn remove_stream(client: &Client, stream: &str) -> Result<(), Error> {
    client.delete_stream().stream_name(stream).send().await?;

    println!("Deleted stream.");

    Ok(())
}
```

- Einzelheiten zur API finden Sie [DeleteStream](#) in der API-Referenz zum AWS SDK für Rust.

DescribeStream

Das folgende Codebeispiel zeigt, wie man es benutzt `DescribeStream`.

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}

```

- Einzelheiten zur API finden Sie [DescribeStream](#) in der API-Referenz zum AWS SDK für Rust.

ListStreams

Das folgende Codebeispiel zeigt, wie man es benutzt `ListStreams`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

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(())
}

```

- Einzelheiten zur API finden Sie [ListStreams](#) in der API-Referenz zum AWS SDK für Rust.

PutRecord

Das folgende Codebeispiel zeigt, wie man es benutzt `PutRecord`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [PutRecord](#) in der API-Referenz zum AWS SDK für Rust.

Serverless-Beispiele

Aufrufen einer Lambda-Funktion über einen Kinesis-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Datensätzen aus einem Kinesis-Stream ausgelöst wird. Die Funktion ruft die Kinesis-Nutzlast ab, dekodiert von Base64 und protokolliert den Datensatzinhalt.

SDK für Rust

Note

Es gibt noch mehr dazu. [GitHub](#) Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Nutzen eines Kinesis-Ereignisses mit Lambda unter Verwendung von 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
}

```

Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem Kinesis-Auslöser

Das folgende Codebeispiel zeigt, wie eine partielle Batch-Antwort für Lambda-Funktionen implementiert wird, die Ereignisse aus einem Kinesis-Stream empfangen. Die Funktion meldet die Batch-Elementfehler in der Antwort und signalisiert Lambda, diese Nachrichten später erneut zu versuchen.

SDK für Rust

Note

Es gibt noch mehr dazu. GitHub Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Melden von Fehlern bei Kinesis-Batchelementen mit Lambda unter Verwendung von 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 Beispiele mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie das AWS SDK für Rust mit verwenden AWS KMS.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

CreateKey

Das folgende Codebeispiel zeigt die Verwendung `CreateKey`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [CreateKey](#) in der API-Referenz zum AWS SDK für Rust.

Decrypt

Das folgende Codebeispiel zeigt, wie man es benutzt `Decrypt`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie unter [Decrypt](#) in AWS SDK for Rust API-Referenz.

Encrypt

Das folgende Codebeispiel zeigt die Verwendung `Encrypt`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie in der API-Referenz zum [Verschlüsseln](#) im AWS SDK für Rust.

GenerateDataKey

Das folgende Codebeispiel zeigt die Verwendung `GenerateDataKey`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [GenerateDataKey](#) in der API-Referenz zum AWS SDK für Rust.

GenerateDataKeyWithoutPlaintext

Das folgende Codebeispiel zeigt, wie man es benutzt `GenerateDataKeyWithoutPlaintext`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [GenerateDataKeyWithoutPlaintext](#) in der API-Referenz zum AWS SDK für Rust.

GenerateRandom

Das folgende Codebeispiel zeigt, wie man es benutzt `GenerateRandom`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [GenerateRandom](#) in der API-Referenz zum AWS SDK für Rust.

ListKeys

Das folgende Codebeispiel zeigt, wie man es benutzt `ListKeys`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [ListKeys](#) in der API-Referenz zum AWS SDK für Rust.

ReEncrypt

Das folgende Codebeispiel zeigt, wie man es benutzt `ReEncrypt`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```


- Einzelheiten zur API finden Sie [ReEncrypt](#) in der API-Referenz zum AWS SDK für Rust.

Lambda-Beispiele mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe des AWS SDK für Rust mit Lambda Aktionen ausführen und allgemeine Szenarien implementieren.

Bei Grundlagen handelt es sich um Code-Beispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Szenarien sind Code-Beispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Services aufrufen oder mit anderen AWS-Services kombinieren.

AWS Community-Beiträge sind Beispiele, die von mehreren AWS Teams erstellt wurden und verwaltet werden. Um Feedback zu geben, verwende den Mechanismus, der in den verlinkten Repositorien zur Verfügung steht.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Grundlagen](#)
- [Aktionen](#)
- [Szenarien](#)
- [Serverless-Beispiele](#)
- [AWS Beiträge der Gemeinschaft](#)

Grundlagen

Erlernen der Grundlagen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie eine IAM-Rolle und eine Lambda-Funktion und laden Sie den Handlercode hoch.

- Rufen Sie die Funktion mit einem einzigen Parameter auf und erhalten Sie Ergebnisse.
- Aktualisieren Sie den Funktionscode und konfigurieren Sie mit einer Umgebungsvariablen.
- Rufen Sie die Funktion mit neuen Parametern auf und erhalten Sie Ergebnisse. Zeigt das zurückgegebene Ausführungsprotokoll an.
- Listen Sie die Funktionen für Ihr Konto auf und bereinigen Sie dann die Ressourcen.

Weitere Informationen zur Verwendung von Lambda finden Sie unter [Erstellen einer Lambda-Funktion mit der Konsole](#).

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Die Datei „Cargo.toml“ mit in diesem Szenario verwendeten Abhängigkeiten.

```
[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"
```

Eine Sammlung von Hilfsprogrammen, die das Aufrufen von Lambda für dieses Szenario optimieren. Diese Datei ist „src/ations.rs“ in der Kiste.

```
use anyhow::anyhow;
use aws_sdk_iam::operation::{create_role::CreateRoleError,
    delete_role::DeleteRoleOutput};
use aws_sdk_lambda::{
    operation::{
        delete_function::DeleteFunctionOutput, get_function::GetFunctionOutput,
        invoke::InvokeOutput, list_functions::ListFunctionsOutput,
        update_function_code::UpdateFunctionCodeOutput,
        update_function_configuration::UpdateFunctionConfigurationOutput,
    },
    primitives::ByteStream,
    types::{Environment, FunctionCode, LastUpdateStatus, State},
};
use aws_sdk_s3::{
    error::ErrorMetadata,
    operation::{delete_bucket::DeleteBucketOutput,
        delete_object::DeleteObjectOutput},
    types::CreateBucketConfiguration,
};
use aws_smithy_types::Blob;
use serde::{ser::SerializeMap, Serialize};
use std::{fmt::Display, path::PathBuf, str::FromStr, time::Duration};
use tracing::{debug, info, warn};

/* Operation describes */
#[derive(Clone, Copy, Debug, Serialize)]
pub enum Operation {
    #[serde(rename = "plus")]
    Plus,
    #[serde(rename = "minus")]
    Minus,
    #[serde(rename = "times")]
    Times,
```

```

    #[serde(rename = "divided-by")]
    DividedBy,
}

impl FromStr for Operation {
    type Err = anyhow::Error;

    fn from_str(s: &str) -> Result<Self, Self::Err> {
        match s {
            "plus" => Ok(Operation::Plus),
            "minus" => Ok(Operation::Minus),
            "times" => Ok(Operation::Times),
            "divided-by" => Ok(Operation::DividedBy),
            _ => Err(anyhow!("Unknown operation {s}")),
        }
    }
}

impl Display for Operation {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        match self {
            Operation::Plus => write!(f, "plus"),
            Operation::Minus => write!(f, "minus"),
            Operation::Times => write!(f, "times"),
            Operation::DividedBy => write!(f, "divided-by"),
        }
    }
}

/**
 * InvokeArgs will be serialized as JSON and sent to the AWS Lambda handler.
 */
#[derive(Debug)]
pub enum InvokeArgs {
    Increment(i32),
    Arithmetic(Operation, i32, i32),
}

impl Serialize for InvokeArgs {
    fn serialize<S>(&self, serializer: S) -> Result<S::Ok, S::Error>
    where
        S: serde::Serializer,
    {
        match self {

```

```

        InvokeArgs::Increment(i) => serializer.serialize_i32(*i),
        InvokeArgs::Arithmetic(o, i, j) => {
            let mut map: S::SerializeMap = serializer.serialize_map(Some(3))?;
            map.serialize_key(&"op".to_string())?;
            map.serialize_value(&o.to_string())?;
            map.serialize_key(&"i".to_string())?;
            map.serialize_value(&i)?;
            map.serialize_key(&"j".to_string())?;
            map.serialize_value(&j)?;
            map.end()
        }
    }
}

/** A policy document allowing Lambda to execute this function on the account's
    behalf. */
const ROLE_POLICY_DOCUMENT: &str = r#{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": { "Service": "lambda.amazonaws.com" },
            "Action": "sts:AssumeRole"
        }
    ]
}";

/**
 * A LambdaManager gathers all the resources necessary to run the Lambda example
 * scenario.
 * This includes instantiated aws_sdk clients and details of resource names.
 */
pub struct LambdaManager {
    iam_client: aws_sdk_iam::Client,
    lambda_client: aws_sdk_lambda::Client,
    s3_client: aws_sdk_s3::Client,
    lambda_name: String,
    role_name: String,
    bucket: String,
    own_bucket: bool,
}

```

```

// These unit type structs provide nominal typing on top of String parameters for
LambdaManager::new
pub struct LambdaName(pub String);
pub struct RoleName(pub String);
pub struct Bucket(pub String);
pub struct OwnBucket(pub bool);

impl LambdaManager {
    pub fn new(
        iam_client: aws_sdk_iam::Client,
        lambda_client: aws_sdk_lambda::Client,
        s3_client: aws_sdk_s3::Client,
        lambda_name: LambdaName,
        role_name: RoleName,
        bucket: Bucket,
        own_bucket: OwnBucket,
    ) -> Self {
        Self {
            iam_client,
            lambda_client,
            s3_client,
            lambda_name: lambda_name.0,
            role_name: role_name.0,
            bucket: bucket.0,
            own_bucket: own_bucket.0,
        }
    }

    /**
     * Load the AWS configuration from the environment.
     * Look up lambda_name and bucket if none are given, or generate a random name
if not present in the environment.
     * If the bucket name is provided, the caller needs to have created the bucket.
     * If the bucket name is generated, it will be created.
     */
    pub async fn load_from_env(lambda_name: Option<String>, bucket: Option<String>)
-> Self {
        let sdk_config = aws_config::load_from_env().await;
        let lambda_name = LambdaName(lambda_name.unwrap_or_else(|| {
            std::env::var("LAMBDA_NAME").unwrap_or_else(|_|
"rust_lambda_example".to_string())
        }));
        let role_name = RoleName(format!("{}_role", lambda_name.0));
        let (bucket, own_bucket) =

```

```
    match bucket {
        Some(bucket) => (Bucket(bucket), false),
        None => (
            Bucket(std::env::var("LAMBDA_BUCKET").unwrap_or_else(|_| {
                format!("rust-lambda-example-{}", uuid::Uuid::new_v4())
            })),
            true,
        ),
    };

let s3_client = aws_sdk_s3::Client::new(&sdk_config);

if own_bucket {
    info!("Creating bucket for demo: {}", bucket.0);
    s3_client
        .create_bucket()
        .bucket(bucket.0.clone())
        .create_bucket_configuration(
            CreateBucketConfiguration::builder()

.location_constraint(aws_sdk_s3::types::BucketLocationConstraint::from(
                sdk_config.region().unwrap().as_ref(),
            ))
            .build(),
        )
        .send()
        .await
        .unwrap();
}

Self::new(
    aws_sdk_iam::Client::new(&sdk_config),
    aws_sdk_lambda::Client::new(&sdk_config),
    s3_client,
    lambda_name,
    role_name,
    bucket,
    OwnBucket(own_bucket),
)
}

/**
 * Upload function code from a path to a zip file.
 * The zip file must have an AL2 Linux-compatible binary called `bootstrap`.
```

```

    * The easiest way to create such a zip is to use `cargo lambda build --output-
format Zip`.
    */
    async fn prepare_function(
        &self,
        zip_file: PathBuf,
        key: Option<String>,
    ) -> Result<FunctionCode, anyhow::Error> {
        let body = ByteStream::from_path(zip_file).await?;

        let key = key.unwrap_or_else(|| format!("{}", self.lambda_name));

        info!("Uploading function code to s3://{}/{}", self.bucket, key);
        let _ = self
            .s3_client
            .put_object()
            .bucket(self.bucket.clone())
            .key(key.clone())
            .body(body)
            .send()
            .await?;

        Ok(FunctionCode::builder()
            .s3_bucket(self.bucket.clone())
            .s3_key(key)
            .build())
    }

    /**
    * Create a function, uploading from a zip file.
    */
    pub async fn create_function(&self, zip_file: PathBuf) -> Result<String,
anyhow::Error> {
        let code = self.prepare_function(zip_file, None).await?;

        let key = code.s3_key().unwrap().to_string();

        let role = self.create_role().await.map_err(|e| anyhow!(e))?;

        info!("Created iam role, waiting 15s for it to become active");
        tokio::time::sleep(Duration::from_secs(15)).await;

        info!("Creating lambda function {}", self.lambda_name);
        let _ = self

```



```

        .lambda_client
        .create_function()
        .function_name(self.lambda_name.clone())
        .code(code)
        .role(role.arn())
        .runtime(aws_sdk_lambda::types::Runtime::Provided2)
        .handler("_unused")
        .send()
        .await
        .map_err( anyhow::Error::from )?;

    self.wait_for_function_ready().await?;

    self.lambda_client
        .publish_version()
        .function_name(self.lambda_name.clone())
        .send()
        .await?;

    Ok(key)
}

/**
 * Create an IAM execution role for the managed Lambda function.
 * If the role already exists, use that instead.
 */
async fn create_role(&self) -> Result<aws_sdk_iam::types::Role, CreateRoleError>
{
    info!("Creating execution role for function");
    let get_role = self
        .iam_client
        .get_role()
        .role_name(self.role_name.clone())
        .send()
        .await;
    if let Ok(get_role) = get_role {
        if let Some(role) = get_role.role {
            return Ok(role);
        }
    }

    let create_role = self
        .iam_client
        .create_role()

```

```

        .role_name(self.role_name.clone())
        .assume_role_policy_document(ROLE_POLICY_DOCUMENT)
        .send()
        .await;

    match create_role {
        Ok(create_role) => match create_role.role {
            Some(role) => Ok(role),
            None => Err(CreateRoleError::generic(
                ErrorMetadata::builder()
                    .message("CreateRole returned empty success")
                    .build(),
            )),
        },
        Err(err) => Err(err.into_service_error()),
    }
}

/**
 * Poll `is_function_ready` with a 1-second delay. It returns when the function
 * is ready or when there's an error checking the function's state.
 */
pub async fn wait_for_function_ready(&self) -> Result<(), anyhow::Error> {
    info!("Waiting for function");
    while !self.is_function_ready(None).await? {
        info!("Function is not ready, sleeping 1s");
        tokio::time::sleep(Duration::from_secs(1)).await;
    }
    Ok(())
}

/**
 * Check if a Lambda function is ready to be invoked.
 * A Lambda function is ready for this scenario when its state is active and its
 * LastUpdateStatus is Successful.
 * Additionally, if a sha256 is provided, the function must have that as its
 * current code hash.
 * Any missing properties or failed requests will be reported as an Err.
 */
async fn is_function_ready(
    &self,
    expected_code_sha256: Option<&str>,
) -> Result<bool, anyhow::Error> {
    match self.get_function().await {

```

```

Ok(func) => {
    if let Some(config) = func.configuration() {
        if let Some(state) = config.state() {
            info!(?state, "Checking if function is active");
            if !matches!(state, State::Active) {
                return Ok(false);
            }
        }
    }
    match config.last_update_status() {
        Some(last_update_status) => {
            info!(?last_update_status, "Checking if function is
ready");

            match last_update_status {
                LastUpdateStatus::Successful => {
                    // continue
                }
                LastUpdateStatus::Failed |
LastUpdateStatus::InProgress => {
                    return Ok(false);
                }
                unknown => {
                    warn!(
                        status_variant = unknown.as_str(),
                        "LastUpdateStatus unknown"
                    );
                    return Err(anyhow!(
                        "Unknown LastUpdateStatus, fn config is
{config:?}"
                    ));
                }
            }
        }
        None => {
            warn!("Missing last update status");
            return Ok(false);
        }
    };
    if expected_code_sha256.is_none() {
        return Ok(true);
    }
    if let Some(code_sha256) = config.code_sha256() {
        return Ok(code_sha256 ==
expected_code_sha256.unwrap_or_default());
    }
}

```

```
    }
  }
  Err(e) => {
    warn!(?e, "Could not get function while waiting");
  }
}
Ok(false)
}

/** Get the Lambda function with this Manager's name. */
pub async fn get_function(&self) -> Result<GetFunctionOutput, anyhow::Error> {
  info!("Getting lambda function");
  self.lambda_client
    .get_function()
    .function_name(self.lambda_name.clone())
    .send()
    .await
    .map_err(anyhow::Error::from)
}

/** List all Lambda functions in the current Region. */
pub async fn list_functions(&self) -> Result<ListFunctionsOutput, anyhow::Error>
{
  info!("Listing lambda functions");
  self.lambda_client
    .list_functions()
    .send()
    .await
    .map_err(anyhow::Error::from)
}

/** Invoke the lambda function using calculator InvokeArgs. */
pub async fn invoke(&self, args: InvokeArgs) -> Result<InvokeOutput,
anyhow::Error> {
  info!(?args, "Invoking {}", self.lambda_name);
  let payload = serde_json::to_string(&args)?;
  debug!(?payload, "Sending payload");
  self.lambda_client
    .invoke()
    .function_name(self.lambda_name.clone())
    .payload(Blob::new(payload))
    .send()
    .await
    .map_err(anyhow::Error::from)
}
```

```
}

/** Given a Path to a zip file, update the function's code and wait for the
update to finish. */
pub async fn update_function_code(
    &self,
    zip_file: PathBuf,
    key: String,
) -> Result<UpdateFunctionCodeOutput, anyhow::Error> {
    let function_code = self.prepare_function(zip_file, Some(key)).await?;

    info!("Updating code for {}", self.lambda_name);
    let update = self
        .lambda_client
        .update_function_code()
        .function_name(self.lambda_name.clone())
        .s3_bucket(self.bucket.clone())
        .s3_key(function_code.s3_key().unwrap().to_string())
        .send()
        .await
        .map_err(anyhow::Error::from)?;

    self.wait_for_function_ready().await?;

    Ok(update)
}

/** Update the environment for a function. */
pub async fn update_function_configuration(
    &self,
    environment: Environment,
) -> Result<UpdateFunctionConfigurationOutput, anyhow::Error> {
    info!(
        ?environment,
        "Updating environment for {}", self.lambda_name
    );
    let updated = self
        .lambda_client
        .update_function_configuration()
        .function_name(self.lambda_name.clone())
        .environment(environment)
        .send()
        .await
        .map_err(anyhow::Error::from)?;
```

```
        self.wait_for_function_ready().await?;

        Ok(updated)
    }

    /** Delete a function and its role, and if possible or necessary, its associated
code object and bucket. */
    pub async fn delete_function(
        &self,
        location: Option<String>,
    ) -> (
        Result<DeleteFunctionOutput, anyhow::Error>,
        Result<DeleteRoleOutput, anyhow::Error>,
        Option<Result<DeleteObjectOutput, anyhow::Error>>,
    ) {
        info!("Deleting lambda function {}", self.lambda_name);
        let delete_function = self
            .lambda_client
            .delete_function()
            .function_name(self.lambda_name.clone())
            .send()
            .await
            .map_err(anyhow::Error::from);

        info!("Deleting iam role {}", self.role_name);
        let delete_role = self
            .iam_client
            .delete_role()
            .role_name(self.role_name.clone())
            .send()
            .await
            .map_err(anyhow::Error::from);

        let delete_object: Option<Result<DeleteObjectOutput, anyhow::Error>> =
            if let Some(location) = location {
                info!("Deleting object {location}");
                Some(
                    self.s3_client
                        .delete_object()
                        .bucket(self.bucket.clone())
                        .key(location)
                        .send()
                        .await
                )
            } else {
                None
            }
    }
}
```

```

        .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(),
        );
    }
}

```

Eine Binärdatei, um das Szenario vom Anfang bis Ende auszuführen, wobei Befehlszeilen-Flags verwendet werden, um einige Verhaltensweisen zu steuern. Diese Datei ist `src/bin/scenario.rs` in der Kiste.

```

/*
## Service actions

Service actions wrap the SDK call, taking a client and any specific parameters
necessary for the call.

* CreateFunction
* GetFunction
* ListFunctions
* Invoke

```



```
* UpdateFunctionCode
* UpdateFunctionConfiguration
* DeleteFunction
```

Scenario

A scenario runs at a command prompt and prints output to the user on the result of each service action. A scenario can run in one of two ways: straight through, printing out progress as it goes, or as an interactive question/answer script.

Getting started with functions

Use an SDK to manage AWS Lambda functions: create a function, invoke it, update its code, invoke it again, view its output and logs, and delete it.

This scenario uses two Lambda handlers:

Note: Handlers don't use AWS SDK API calls.

The increment handler is straightforward:

1. It accepts a number, increments it, and returns the new value.
2. It performs simple logging of the result.

The arithmetic handler is more complex:

1. It accepts a set of actions ['plus', 'minus', 'times', 'divided-by'] and two numbers, and returns the result of the calculation.
2. It uses an environment variable to control log level (such as DEBUG, INFO, WARNING, ERROR).

It logs a few things at different levels, such as:

- * DEBUG: Full event data.
- * INFO: The calculation result.
- * WARN~ING~: When a divide by zero error occurs.
- * This will be the typical `RUST_LOG` variable.

The steps of the scenario are:

1. Create an AWS Identity and Access Management (IAM) role that meets the following requirements:
 - * Has an `assume_role` policy that grants 'lambda.amazonaws.com' the 'sts:AssumeRole' action.
 - * Attaches the 'arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole' managed role.
 - * You must wait for ~10 seconds after the role is created before you can use it!

2. Create a function (CreateFunction) for the increment handler by packaging it as a zip and doing one of the following:
 - * Adding it with CreateFunction Code.ZipFile.
 - * --or--
 - * Uploading it to Amazon Simple Storage Service (Amazon S3) and adding it with CreateFunction Code.S3Bucket/S3Key.
 - * Note: Zipping the file does not have to be done in code.
 - * If you have a waiter, use it to wait until the function is active. Otherwise, call GetFunction until State is Active.
3. Invoke the function with a number and print the result.
4. Update the function (UpdateFunctionCode) to the arithmetic handler by packaging it as a zip and doing one of the following:
 - * Adding it with UpdateFunctionCode ZipFile.
 - * --or--
 - * Uploading it to Amazon S3 and adding it with UpdateFunctionCode S3Bucket/S3Key.
5. Call GetFunction until Configuration.LastUpdateStatus is 'Successful' (or 'Failed').
6. Update the environment variable by calling UpdateFunctionConfiguration and pass it a log level, such as:
 - * Environment={'Variables': {'RUST_LOG': 'TRACE'}}
7. Invoke the function with an action from the list and a couple of values. Include LogType='Tail' to get logs in the result. Print the result of the calculation and the log.
8. [Optional] Invoke the function to provoke a divide-by-zero error and show the log result.
9. List all functions for the account, using pagination (ListFunctions).
10. Delete the function (DeleteFunction).
11. Delete the role.

Each step should use the function created in Service Actions to abstract calling the SDK.

```
*/
```

```
use aws_sdk_lambda::{operation::invoke::InvokeOutput, types::Environment};
use clap::Parser;
use std::{collections::HashMap, path::PathBuf};
use tracing::{debug, info, warn};
use tracing_subscriber::EnvFilter;

use lambda_code_examples::actions::{
    InvokeArgs::{Arithmetic, Increment},
    LambdaManager, Operation,
};
```

```
#[derive(Debug, Parser)]
pub struct Opt {
    /// The AWS Region.
    #[structopt(short, long)]
    pub region: Option<String>,

    // The bucket to use for the FunctionCode.
    #[structopt(short, long)]
    pub bucket: Option<String>,

    // The name of the Lambda function.
    #[structopt(short, long)]
    pub lambda_name: Option<String>,

    // The number to increment.
    #[structopt(short, long, default_value = "12")]
    pub inc: i32,

    // The left operand.
    #[structopt(long, default_value = "19")]
    pub num_a: i32,

    // The right operand.
    #[structopt(long, default_value = "23")]
    pub num_b: i32,

    // The arithmetic operation.
    #[structopt(short, long, default_value = "plus")]
    pub operation: Operation,

    #[structopt(long)]
    pub cleanup: Option<bool>,

    #[structopt(long)]
    pub no_cleanup: Option<bool>,
}

fn code_path(lambda: &str) -> PathBuf {
    PathBuf::from(format!("../target/lambda/{lambda}/bootstrap.zip"))
}

fn log_invoke_output(invoker: &InvokeOutput, message: &str) {
    if let Some(payload) = invoker.payload().cloned() {
```

```
        let payload = String::from_utf8(payload.into_inner());
        info!(?payload, message);
    } else {
        info!("Could not extract payload")
    }
    if let Some(logs) = invoke.log_result() {
        debug!(?logs, "Invoked function logs")
    } else {
        debug!("Invoked function had no logs")
    }
}

async fn main_block(
    opt: &Opt,
    manager: &LambdaManager,
    code_location: String,
) -> Result<(), anyhow::Error> {
    let invoke = manager.invoke(Increment(opt.inc)).await?;
    log_invoke_output(&invoke, "Invoked function configured as increment");

    let update_code = manager
        .update_function_code(code_path("arithmetic"), code_location.clone())
        .await?;

    let code_sha256 = update_code.code_sha256().unwrap_or("Unknown SHA");
    info!(?code_sha256, "Updated function code with arithmetic.zip");

    let arithmetic_args = Arithmetic(opt.operation, opt.num_a, opt.num_b);
    let invoke = manager.invoke(arithmetic_args).await?;
    log_invoke_output(&invoke, "Invoked function configured as arithmetic");

    let update = manager
        .update_function_configuration(
            Environment::builder()
                .set_variables(Some(HashMap::from([
                    "RUST_LOG".to_string(),
                    "trace".to_string(),
                ])))
                .build(),
        )
        .await?;
    let updated_environment = update.environment();
    info!(?updated_environment, "Updated function configuration");
}
```

```

    let invoke = manager
        .invoke(Arithmetic(opt.operation, opt.num_a, opt.num_b))
        .await?;
    log_invoke_output(
        &invoke,
        "Invoked function configured as arithmetic with increased logging",
    );

    let invoke = manager
        .invoke(Arithmetic(Operation::DividedBy, opt.num_a, 0))
        .await?;
    log_invoke_output(
        &invoke,
        "Invoked function configured as arithmetic with divide by zero",
    );

    Ok::<(), anyhow::Error>(( ))
}

#[tokio::main]
async fn main() {
    tracing_subscriber::fmt()
        .without_time()
        .with_file(true)
        .with_line_number(true)
        .with_env_filter(EnvFilter::from_default_env())
        .init();

    let opt = Opt::parse();
    let manager = LambdaManager::load_from_env(opt.lambda_name.clone(),
    opt.bucket.clone()).await;

    let key = match manager.create_function(code_path("increment")).await {
        Ok(init) => {
            info!(?init, "Created function, initially with increment.zip");
            let run_block = main_block(&opt, &manager, init.clone()).await;
            info!(?run_block, "Finished running example, cleaning up");
            Some(init)
        }
        Err(err) => {
            warn!(?err, "Error happened when initializing function");
            None
        }
    };
};

```

```
    if Some(false) == opt.cleanup || Some(true) == opt.no_cleanup {
        info!("Skipping cleanup")
    } else {
        let delete = manager.cleanup(key).await;
        info!(?delete, "Deleted function & cleaned up resources");
    }
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der API-Referenz zu AWS - SDK für Rust.
 - [CreateFunction](#)
 - [DeleteFunction](#)
 - [GetFunction](#)
 - [Aufrufen](#)
 - [ListFunctions](#)
 - [UpdateFunctionCode](#)
 - [UpdateFunctionConfiguration](#)

Aktionen

CreateFunction

Das folgende Codebeispiel zeigt, wie man es benutzt. CreateFunction

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/**
 * Create a function, uploading from a zip file.
 */
```

```

    pub async fn create_function(&self, zip_file: PathBuf) -> Result<String,
anyhow::Error> {
        let code = self.prepare_function(zip_file, None).await?;

        let key = code.s3_key().unwrap().to_string();

        let role = self.create_role().await.map_err(|e| anyhow!(e))?;

        info!("Created iam role, waiting 15s for it to become active");
        tokio::time::sleep(Duration::from_secs(15)).await;

        info!("Creating lambda function {}", self.lambda_name);
        let _ = self
            .lambda_client
            .create_function()
            .function_name(self.lambda_name.clone())
            .code(code)
            .role(role.arn())
            .runtime(aws_sdk_lambda::types::Runtime::Providedal2)
            .handler("_unused")
            .send()
            .await
            .map_err(anyhow::Error::from)?;

        self.wait_for_function_ready().await?;

        self.lambda_client
            .publish_version()
            .function_name(self.lambda_name.clone())
            .send()
            .await?;

        Ok(key)
    }

    /**
     * Upload function code from a path to a zip file.
     * The zip file must have an AL2 Linux-compatible binary called `bootstrap`.
     * The easiest way to create such a zip is to use `cargo lambda build --output-
format Zip`.
     */
    async fn prepare_function(
        &self,
        zip_file: PathBuf,

```

```

    key: Option<String>,
) -> Result<FunctionCode, anyhow::Error> {
    let body = ByteStream::from_path(zip_file).await?;

    let key = key.unwrap_or_else(|| format!("{}_code", self.lambda_name));

    info!("Uploading function code to s3://{}/{}", self.bucket, key);
    let _ = self
        .s3_client
        .put_object()
        .bucket(self.bucket.clone())
        .key(key.clone())
        .body(body)
        .send()
        .await?;

    Ok(FunctionCode::builder()
        .s3_bucket(self.bucket.clone())
        .s3_key(key)
        .build())
}

```

- Einzelheiten zur API finden Sie [CreateFunction](#) in der API-Referenz zum AWS SDK für Rust.

DeleteFunction

Das folgende Codebeispiel zeigt, wie man es benutzt `DeleteFunction`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

/** 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)
}
```

- Einzelheiten zur API finden Sie [DeleteFunction](#) in der API-Referenz zum AWS SDK für Rust.

GetFunction

Das folgende Codebeispiel zeigt, wie man es benutzt `GetFunction`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/** 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)
}
```

- Einzelheiten zur API finden Sie [GetFunction](#) in der API-Referenz zum AWS SDK für Rust.

Invoke

Das folgende Codebeispiel zeigt, wie man es benutzt `Invoke`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/** Invoke the lambda function using calculator InvokeArgs. */
pub async fn invoke(&self, args: InvokeArgs) -> Result<InvokeOutput,
anyhow::Error> {
    info!(?args, "Invoking {}", self.lambda_name);
    let payload = serde_json::to_string(&args)?;
    debug!(?payload, "Sending payload");
    self.lambda_client
        .invoke()
        .function_name(self.lambda_name.clone())
        .payload(Blob::new(payload))
        .send()
        .await
        .map_err(anyhow::Error::from)
}


fn log_invoke_output(invoker: &InvokeOutput, message: &str) {
    if let Some(payload) = invoker.payload().cloned() {
        let payload = String::from_utf8(payload.into_inner());
        info!(?payload, message);
    } else {
        info!("Could not extract payload")
    }
    if let Some(logs) = invoker.log_result() {
        debug!(?logs, "Invoked function logs")
    } else {
        debug!("Invoked function had no logs")
    }
}
```

- Weitere API-Informationen finden Sie unter [Aufrufen](#) in der API-Referenz zum AWS -SDK für Rust.

ListFunctions

Das folgende Codebeispiel zeigt die Verwendung `ListFunctions`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/** 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)
}
```

- Einzelheiten zur API finden Sie [ListFunctions](#) in der API-Referenz zum AWS SDK für Rust.

UpdateFunctionCode

Das folgende Codebeispiel zeigt, wie man es benutzt `UpdateFunctionCode`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/** Given a Path to a zip file, update the function's code and wait for the
update to finish. */
pub async fn update_function_code(
    &self,
    zip_file: PathBuf,
    key: String,
```

```

) -> Result<UpdateFunctionCodeOutput, anyhow::Error> {
    let function_code = self.prepare_function(zip_file, Some(key)).await?;

    info!("Updating code for {}", self.lambda_name);
    let update = self
        .lambda_client
        .update_function_code()
        .function_name(self.lambda_name.clone())
        .s3_bucket(self.bucket.clone())
        .s3_key(function_code.s3_key().unwrap().to_string())
        .send()
        .await
        .map_err(anyhow::Error::from)?;

    self.wait_for_function_ready().await?;

    Ok(update)
}

/**
 * Upload function code from a path to a zip file.
 * The zip file must have an AL2 Linux-compatible binary called `bootstrap`.
 * The easiest way to create such a zip is to use `cargo lambda build --output-format Zip`.
 */
async fn prepare_function(
    &self,
    zip_file: PathBuf,
    key: Option<String>,
) -> Result<FunctionCode, anyhow::Error> {
    let body = ByteStream::from_path(zip_file).await?;

    let key = key.unwrap_or_else(|| format!("{}_code", self.lambda_name));

    info!("Uploading function code to s3://{}/{}", self.bucket, key);
    let _ = self
        .s3_client
        .put_object()
        .bucket(self.bucket.clone())
        .key(key.clone())
        .body(body)
        .send()
        .await?;

```

```

Ok(FunctionCode::builder()
    .s3_bucket(self.bucket.clone())
    .s3_key(key)
    .build())
}

```

- Einzelheiten zur API finden Sie [UpdateFunctionCode](#) in der API-Referenz zum AWS SDK für Rust.

UpdateFunctionConfiguration

Das folgende Codebeispiel zeigt, wie man es benutzt `UpdateFunctionConfiguration`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

/** 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)  
}
```

- Einzelheiten zur API finden Sie [UpdateFunctionConfiguration](#) in der API-Referenz zum AWS SDK für Rust.

Szenarien

Erstellen einer Serverless-Anwendung zur Verwaltung von Fotos

Das folgende Codebeispiel zeigt, wie eine Serverless-Anwendung erstellt wird, mit der Benutzer Fotos mithilfe von Labels erstellen können.

SDK für Rust

Zeigt, wie eine Anwendung zur Verwaltung von Fotobeständen entwickelt wird, die mithilfe von Amazon Rekognition Labels in Bildern erkennt und sie für einen späteren Abruf speichert.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

Einen tiefen Einblick in den Ursprung dieses Beispiels finden Sie im Beitrag in der [AWS - Community](#).

In diesem Beispiel verwendete Dienste

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

Serverless-Beispiele

Herstellen einer Verbindung mit einer Amazon-RDS-Datenbank in einer Lambda-Funktion

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die eine Verbindung zu einer RDS-Datenbank herstellt. Die Funktion stellt eine einfache Datenbankanfrage und gibt das Ergebnis zurück.

SDK für Rust

Note

Es gibt noch mehr dazu. [GitHub](#) Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Herstellen einer Verbindung zu einer Amazon-RDS-Datenbank in einer Lambda-Funktion mit 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());

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}")
    })))
}
```

Aufrufen einer Lambda-Funktion über einen Kinesis-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Datensätzen aus einem Kinesis-Stream ausgelöst wird. Die Funktion ruft die Kinesis-Nutzlast ab, dekodiert von Base64 und protokolliert den Datensatzinhalt.

SDK für Rust

Note

Es gibt noch mehr dazu. [GitHub](#) Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Nutzen eines Kinesis-Ereignisses mit Lambda unter Verwendung von 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
}

```

Aufrufen einer Lambda-Funktion über einen DynamoDB-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Datensätzen aus einem DynamoDB-Stream ausgelöst wird. Die Funktion ruft die DynamoDB-Nutzdaten ab und protokolliert den Inhalt des Datensatzes.

SDK für Rust

Note

Es gibt noch mehr dazu. [GitHub](#) Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Nutzen eines DynamoDB-Ereignisses mit Lambda unter Verwendung von 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(())
}
```

Aufrufen einer Lambda-Funktion über einen Amazon DocumentDB-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Datensätzen aus einem DocumentDB-Änderungsstream ausgelöst wird. Die Funktion ruft die DocumentDB-Nutzdaten ab und protokolliert den Inhalt des Datensatzes.

SDK für Rust

Note

Es gibt noch mehr dazu. [GitHub](#) Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Nutzen eines Amazon DocumentDB-Ereignisses mit Lambda unter Verwendung von 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(())
}
}
```

Aufrufen einer Lambda-Funktion über einen Amazon-MSK-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Datensätzen aus einem Amazon MSK-Cluster ausgelöst wird. Die Funktion ruft die MSK-Nutzdaten ab und protokolliert den Inhalt des Datensatzes.

SDK für Rust

Note

Es gibt noch mehr dazu. [GitHub](#) Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Ein Amazon MSK-Ereignis mit Lambda mithilfe von Rust konsumieren.

```
use aws_lambda_events::event::kafka::KafkaEvent;
use lambda_runtime::{run, service_fn, tracing, Error, LambdaEvent};
use base64::prelude::*;
use serde_json::Value;
use tracing::info;

/// Pre-Requisites:
/// 1. Install Cargo Lambda - see https://www.cargo-lambda.info/guide/getting-started.html
/// 2. Add packages tracing, tracing-subscriber, serde_json, base64
///
/// This is the main body for the function.
/// Write your code inside it.
/// There are some code example in the following URLs:
/// - https://github.com/aws-labs/aws-lambda-rust-runtime/tree/main/examples
/// - https://github.com/aws-samples/serverless-rust-demo/
```



```
async fn function_handler(event: LambdaEvent<KafkaEvent>) -> Result<Value, Error> {

    let payload = event.payload.records;

    for (_name, records) in payload.iter() {

        for record in records {

            let record_text = record.value.as_ref().ok_or("Value is None")?;
            info!("Record: {}", &record_text);

            // perform Base64 decoding
            let record_bytes = BASE64_STANDARD.decode(record_text)?;
            let message = std::str::from_utf8(&record_bytes)?;

            info!("Message: {}", message);
        }

    }

    Ok(()).into()
}

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

    // required to enable CloudWatch error logging by the runtime
    tracing::init_default_subscriber();
    info!("Setup CW subscriber!");

    run(service_fn(function_handler)).await
}
```

Aufrufen einer Lambda-Funktion über einen Amazon-S3-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch das Hochladen eines Objekts in einen S3-Bucket ausgelöst wird. Die Funktion ruft den Namen des S3-Buckets sowie den Objektschlüssel aus dem Ereignisparameter ab und ruft die Amazon-S3-API auf, um den Inhaltstyp des Objekts abzurufen und zu protokollieren.

SDK für Rust

 Note

Es gibt noch mehr dazu. [GitHub](#) Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Nutzen eines S3-Ereignisses mit Lambda unter Verwendung von 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(())
}
```

Eine Lambda-Funktion über einen Amazon-SNS-Trigger aufrufen

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Nachrichten von einem SNS-Thema ausgelöst wird. Die Funktion ruft die Nachrichten aus dem Ereignisparameter ab und protokolliert den Inhalt jeder Nachricht.

SDK für Rust

Note

Es gibt noch mehr dazu. [GitHub](#) Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Nutzen eines SNS-Ereignisses mit Lambda unter Verwendung von 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;

// 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
}
```

```
}
```

Aufrufen einer Lambda-Funktion über einen Amazon-SQS-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Nachrichten aus einer SQS-Warteschlange ausgelöst wird. Die Funktion ruft die Nachrichten aus dem Ereignisparameter ab und protokolliert den Inhalt jeder Nachricht.

SDK für Rust

Note

Es gibt noch mehr dazu. [GitHub](#) Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Nutzen eines SQS-Ereignisses mit Lambda unter Verwendung von 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
}

```

Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem Kinesis-Auslöser

Das folgende Codebeispiel zeigt, wie eine partielle Batch-Antwort für Lambda-Funktionen implementiert wird, die Ereignisse aus einem Kinesis-Stream empfangen. Die Funktion meldet die Batch-Elementfehler in der Antwort und signalisiert Lambda, diese Nachrichten später erneut zu versuchen.

SDK für Rust

Note

Es gibt noch mehr dazu. [GitHub](#) Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Melden von Fehlern bei Kinesis-Batchelementen mit Lambda unter Verwendung von 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
}

```

Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem DynamoDB-Auslöser

Das folgende Codebeispiel zeigt, wie eine partielle Batch-Antwort für Lambda-Funktionen implementiert wird, die Ereignisse aus einem DynamoDB-Stream empfangen. Die Funktion meldet die Batch-Elementfehler in der Antwort und signalisiert Lambda, diese Nachrichten später erneut zu versuchen.

SDK für Rust

Note

Es gibt noch mehr dazu. GitHub Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Melden von DynamoDB-Batchelementfehlern mit Lambda unter Verwendung von 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...
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
}
```

Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem Amazon-SQS-Auslöser

Das folgende Codebeispiel zeigt, wie eine partielle Batch-Antwort für Lambda-Funktionen implementiert wird, die Ereignisse aus einer SQS-Warteschlange empfangen. Die Funktion meldet die Batch-Elementfehler in der Antwort und signalisiert Lambda, diese Nachrichten später erneut zu versuchen.

SDK für Rust

Note

Es gibt noch mehr dazu. [GitHub](#) Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Melden von Fehlern bei SQS-Batchelementen mit Lambda unter Verwendung von 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 Beiträge der Gemeinschaft

Erstellen und testen Sie eine serverlose Anwendung

Das folgende Codebeispiel zeigt, wie eine serverlose Anwendung mithilfe von API Gateway mit Lambda und DynamoDB erstellt und getestet wird.

SDK für Rust

Zeigt, wie eine serverlose Anwendung, die aus einem API Gateway mit Lambda und DynamoDB besteht, mithilfe des Rust-SDK erstellt und getestet wird.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter. [GitHub](#)

In diesem Beispiel verwendete Dienste

- API Gateway
- DynamoDB
- Lambda

MediaLive Beispiele mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie das AWS SDK für Rust mit verwenden MediaLive.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

ListInputs

Das folgende Codebeispiel zeigt die VerwendungListInputs.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

MediaLive Geben Sie Ihre Eingabenamen und ARNs die Region an.

```
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(())
}
```

- Einzelheiten zur API finden Sie [ListInputs](#) in der API-Referenz zum AWS SDK für Rust.

MediaPackage Beispiele für die Verwendung von SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie das AWS SDK für Rust mit verwenden MediaPackage.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen


- [Aktionen](#)

Aktionen

ListChannels

Das folgende Codebeispiel zeigt die Verwendung `ListChannels`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Kanal ARNs und Beschreibungen auflisten.

```
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(())
}
```

- Einzelheiten zur API finden Sie [ListChannels](#) in der API-Referenz zum AWS SDK für Rust.

ListOriginEndpoints

Das folgende Codebeispiel zeigt, wie man es benutzt `ListOriginEndpoints`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Listen Sie Ihre Endpunktbeschreibungen auf und URLs.

```
async fn show_endpoints(client: &Client) -> Result<(), Error> {
    let or_endpoints = client.list_origin_endpoints().send().await?;

    println!("Endpoints:");

    for e in or_endpoints.origin_endpoints() {
        let endpoint_url = e.url().unwrap_or_default();
        let endpoint_description = e.description().unwrap_or_default();
        println!(" Description: {}", endpoint_description);
        println!(" URL :          {}", endpoint_url);
        println!();
    }

    Ok(())
}
```

- Einzelheiten zur API finden Sie [ListOriginEndpoints](#) in der API-Referenz zum AWS SDK für Rust.

Amazon MSK-Beispiele mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe des AWS SDK für Rust mit Amazon MSK Aktionen ausführen und allgemeine Szenarien implementieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Serverless-Beispiele](#)

Serverless-Beispiele

Aufrufen einer Lambda-Funktion über einen Amazon-MSK-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Datensätzen aus einem Amazon MSK-Cluster ausgelöst wird. Die Funktion ruft die MSK-Nutzdaten ab und protokolliert den Inhalt des Datensatzes.

SDK für Rust

 Note

Es gibt noch mehr dazu. [GitHub](#) Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Ein Amazon MSK-Ereignis mit Lambda mithilfe von Rust konsumieren.

```
use aws_lambda_events::event::kafka::KafkaEvent;
use lambda_runtime::{run, service_fn, tracing, Error, LambdaEvent};
use base64::prelude::*;
use serde_json::{Value};
use tracing::{info};

/// Pre-Requisites:
/// 1. Install Cargo Lambda - see https://www.cargo-lambda.info/guide/getting-
started.html
/// 2. Add packages tracing, tracing-subscriber, serde_json, base64
///
/// This is the main body for the function.
/// Write your code inside it.
/// There are some code example in the following URLs:
/// - https://github.com/awslabs/aws-lambda-rust-runtime/tree/main/examples
/// - https://github.com/aws-samples/serverless-rust-demo/

async fn function_handler(event: LambdaEvent<KafkaEvent>) -> Result<Value, Error> {

    let payload = event.payload.records;

    for (_name, records) in payload.iter() {

        for record in records {

            let record_text = record.value.as_ref().ok_or("Value is None")?;
            info!("Record: {}", &record_text);

            // perform Base64 decoding
            let record_bytes = BASE64_STANDARD.decode(record_text)?;
            let message = std::str::from_utf8(&record_bytes)?;

            info!("Message: {}", message);
```



```
    }

    }

    Ok(()).into()
}

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

    // required to enable CloudWatch error logging by the runtime
    tracing::init_default_subscriber();
    info!("Setup CW subscriber!");

    run(service_fn(function_handler)).await
}
```

Amazon Polly Polly-Beispiele mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe des AWS SDK für Rust mit Amazon Polly Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Szenarien sind Code-Beispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Services aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)
- [Szenarien](#)

Aktionen

DescribeVoices

Das folgende Codebeispiel zeigt die Verwendung `DescribeVoices`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [DescribeVoices](#) in der API-Referenz zum AWS SDK für Rust.

ListLexicons

Das folgende Codebeispiel zeigt, wie man es benutzt `ListLexicons`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [ListLexicons](#) in der API-Referenz zum AWS SDK für Rust.

PutLexicon

Das folgende Codebeispiel zeigt, wie man es benutzt `PutLexicon`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

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(())
}


```

- Einzelheiten zur API finden Sie [PutLexicon](#) in der API-Referenz zum AWS SDK für Rust.

SynthesizeSpeech

Das folgende Codebeispiel zeigt, wie man es benutzt `SynthesizeSpeech`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [SynthesizeSpeech](#) in der API-Referenz zum AWS SDK für Rust.

Szenarien

Text in Sprache und zurück in Text konvertieren

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Verwenden Sie Amazon Polly, um eine Nur-Text-Eingabedatei (UTF-8) in eine Audiodatei zu synthetisieren.
- Laden Sie die Audiodatei in einen Amazon-S3-Bucket hoch.
- Konvertieren Sie die Audiodatei mit Amazon Transcribe in Text.
- Zeigen Sie den Text an.

SDK für Rust

Verwenden Sie Amazon Polly, um eine Klartext-Eingabedatei (UTF-8) in eine Audiodatei zu synthetisieren, die Audiodatei in einen Amazon-S3-Bucket hochzuladen, diese Audiodatei mit Amazon Transcribe in Text zu konvertieren und den Text anzuzeigen.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- Amazon Polly
- Amazon S3
- Amazon Transcribe

QLDB-Beispiele mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe des AWS SDK für Rust mit QLDB Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

CreateLedger

Das folgende Codebeispiel zeigt die Verwendung `CreateLedger`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [CreateLedger](#) in der API-Referenz zum AWS SDK für Rust.

ListLedgers

Das folgende Codebeispiel zeigt, wie man es benutzt `ListLedgers`.

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [ListLedgers](#) in der API-Referenz zum AWS SDK für Rust.

Amazon RDS-Beispiele mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe des AWS SDK für Rust mit Amazon RDS Aktionen ausführen und allgemeine Szenarien implementieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Themen

- [Serverless-Beispiele](#)

Serverless-Beispiele

Herstellen einer Verbindung mit einer Amazon-RDS-Datenbank in einer Lambda-Funktion

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die eine Verbindung zu einer RDS-Datenbank herstellt. Die Funktion stellt eine einfache Datenbankanfrage und gibt das Ergebnis zurück.

SDK für Rust

Note

Es gibt noch mehr dazu. [GitHub](#) Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Herstellen einer Verbindung zu einer Amazon-RDS-Datenbank in einer Lambda-Funktion mit 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());

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}")
    })))
}
```

Beispiele für Amazon RDS Data Service mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe des AWS SDK für Rust mit Amazon RDS Data Service Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

ExecuteStatement

Das folgende Codebeispiel zeigt die Verwendung `ExecuteStatement`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [ExecuteStatement](#) in der API-Referenz zum AWS SDK für Rust.

Amazon Rekognition Rekognition-Beispiele mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe des AWS SDK für Rust mit Amazon Rekognition Aktionen ausführen und gängige Szenarien implementieren.

Szenarien sind Code-Beispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Services aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Themen

- [Szenarien](#)

Szenarien

Erstellen einer Serverless-Anwendung zur Verwaltung von Fotos

Das folgende Codebeispiel zeigt, wie eine Serverless-Anwendung erstellt wird, mit der Benutzer Fotos mithilfe von Labels erstellen können.

SDK für Rust

Zeigt, wie eine Anwendung zur Verwaltung von Fotobeständen entwickelt wird, die mithilfe von Amazon Rekognition Labels in Bildern erkennt und sie für einen späteren Abruf speichert.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

Einen tiefen Einblick in den Ursprung dieses Beispiels finden Sie im Beitrag in der [AWS - Community](#).

In diesem Beispiel verwendete Dienste

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

Gesichter in einem Bild erkennen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Speichern Sie ein Bild in einem Amazon-S3-Bucket.
- Verwenden Sie Amazon Rekognition, um Gesichtsdetails wie Altersgruppe, Geschlecht und Emotionen (z B. Lächeln) zu erkennen.
- Zeigen Sie diese Details an.

SDK für Rust

Speichern Sie das Bild in einem Amazon-S3-Bucket mit einem uploads-Präfix, verwenden Sie Amazon Rekognition, um Gesichtsdetails wie Altersgruppe, Geschlecht und Emotionen (Lächeln usw.) zu erkennen, und zeigen Sie diese Details an.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- Amazon Rekognition
- Amazon S3

EXIF- und andere Bildinformationen speichern

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Rufen Sie EXIF-Informationen aus einer JPG-, JPEG- oder PNG-Datei ab.
- Laden Sie die Bilddatei in einen Amazon-S3-Bucket hoch.
- Verwenden Sie Amazon Rekognition, um die drei wichtigsten Attribute (Labels) in der Datei zu identifizieren.
- Fügen Sie die EXIF- und Label-Informationen einer Amazon-DynamoDB-Tabelle in der Region hinzu.

SDK für Rust

Rufen Sie EXIF-Informationen aus einer JPG-, JPEG- oder PNG-Datei ab, laden Sie die Bilddatei in einen Amazon-S3-Bucket hoch und identifizieren Sie mit Amazon Rekognition die drei wichtigsten Attribute (Labels in Amazon Rekognition) in der Datei. Fügen Sie die EXIF- und Labelinformationen dann einer Amazon-DynamoDB-Tabelle in der Region hinzu.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- DynamoDB
- Amazon Rekognition
- Amazon S3

Route-53-Beispiele mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe des AWS SDK für Rust mit Route 53 Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

ListHostedZones

Das folgende Codebeispiel zeigt die Verwendung `ListHostedZones`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [ListHostedZones](#) in der API-Referenz zum AWS SDK für Rust.

Amazon S3 S3-Beispiele mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe des AWS SDK für Rust mit Amazon S3 Aktionen ausführen und allgemeine Szenarien implementieren.

Bei Grundlagen handelt es sich um Code-Beispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Szenarien sind Code-Beispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Services aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Erste Schritte

Hello Amazon S3

Die folgenden Codebeispiele veranschaulichen die ersten Schritte mit Amazon S3.

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

/// 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(())
}

```

ExampleError S3-Dienstprogramme.

```

/// 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)
    }
}
}

```

- Einzelheiten zur API finden Sie [ListBuckets](#) in der API-Referenz zum AWS SDK für Rust.

Themen

- [Grundlagen](#)
- [Aktionen](#)
- [Szenarien](#)
- [Serverless-Beispiele](#)


```
#[tokio::main]
async fn main() -> Result<(), S3ExampleError> {
    let region_provider = RegionProviderChain::first_try(Region::new("us-west-2"));
    let region = region_provider.region().await.unwrap();
    let shared_config = aws_config::from_env().region(region_provider).load().await;
    let client = Client::new(&shared_config);
    let bucket_name = format!("amzn-s3-demo-bucket-{}", Uuid::new_v4());
    let file_name = "s3/testfile.txt".to_string();
    let key = "test file key name".to_string();
    let target_key = "target_key".to_string();

    if let Err(e) = run_s3_operations(region, client, bucket_name, file_name, key,
target_key).await
    {
        eprintln!("{:?}", e);
    };

    Ok(())
}

async fn run_s3_operations(
    region: Region,
    client: Client,
    bucket_name: String,
    file_name: String,
    key: String,
    target_key: String,
) -> Result<(), S3ExampleError> {
    s3_code_examples::create_bucket(&client, &bucket_name, &region).await?;
    let run_example: Result<(), S3ExampleError> = (async {
        s3_code_examples::upload_object(&client, &bucket_name, &file_name,
&key).await?;
        let _object = s3_code_examples::download_object(&client, &bucket_name,
&key).await;
        s3_code_examples::copy_object(&client, &bucket_name, &bucket_name, &key,
&target_key)
            .await?;
        s3_code_examples::list_objects(&client, &bucket_name).await?;
        s3_code_examples::clear_bucket(&client, &bucket_name).await?;
        Ok(())
    })
    .await;
    if let Err(err) = run_example {
```

```

        eprintln!("Failed to complete getting-started example: {err:?}");
    }
    s3_code_examples::delete_bucket(&client, &bucket_name).await?;

    Ok(())
}

```

Allgemeine Aktionen, die im Szenario verwendet werden.

```

pub async fn create_bucket(
    client: &aws_sdk_s3::Client,
    bucket_name: &str,
    region: &aws_config::Region,
) -> Result<Option<aws_sdk_s3::operation::create_bucket::CreateBucketOutput>,
S3ExampleError> {
    let constraint =
aws_sdk_s3::types::BucketLocationConstraint::from(region.to_string().as_str());
    let cfg = aws_sdk_s3::types::CreateBucketConfiguration::builder()
        .location_constraint(constraint)
        .build();
    let create = client
        .create_bucket()
        .create_bucket_configuration(cfg)
        .bucket(bucket_name)
        .send()
        .await;

    // BucketAlreadyExists and BucketAlreadyOwnedByYou are not problems for this
    task.
    create.map(Some).or_else(|err| {
        if err
            .as_service_error()
            .map(|se| se.is_bucket_already_exists() ||
se.is_bucket_already_owned_by_you())
            == Some(true)
        {
            Ok(None)
        } else {
            Err(S3ExampleError::from(err))
        }
    })
}

```

```
pub async fn upload_object(
    client: &aws_sdk_s3::Client,
    bucket_name: &str,
    file_name: &str,
    key: &str,
) -> Result<aws_sdk_s3::operation::put_object::PutObjectOutput, S3ExampleError> {
    let body =
aws_sdk_s3::primitives::ByteStream::from_path(std::path::Path::new(file_name)).await;
    client
        .put_object()
        .bucket(bucket_name)
        .key(key)
        .body(body.unwrap())
        .send()
        .await
        .map_err(S3ExampleError::from)
}

pub async fn download_object(
    client: &aws_sdk_s3::Client,
    bucket_name: &str,
    key: &str,
) -> Result<aws_sdk_s3::operation::get_object::GetObjectOutput, S3ExampleError> {
    client
        .get_object()
        .bucket(bucket_name)
        .key(key)
        .send()
        .await
        .map_err(S3ExampleError::from)
}

/// Copy an object from one bucket to another.
pub async fn copy_object(
    client: &aws_sdk_s3::Client,
    source_bucket: &str,
    destination_bucket: &str,
    source_object: &str,
    destination_object: &str,
) -> Result<(), S3ExampleError> {
    let source_key = format!("{source_bucket}/{source_object}");
    let response = client
        .copy_object()
```



```

        .copy_source(&source_key)
        .bucket(destination_bucket)
        .key(destination_object)
        .send()
        .await?;

println!(
    "Copied from {source_key} to {destination_bucket}/{destination_object} with
etag {}",
    response
        .copy_object_result
        .unwrap_or_else(||
aws_sdk_s3::types::CopyObjectResult::builder().build())
        .e_tag()
        .unwrap_or("missing")
);
Ok(())
}

pub async fn list_objects(client: &aws_sdk_s3::Client, bucket: &str) -> Result<(),
S3ExampleError> {
    let mut response = client
        .list_objects_v2()
        .bucket(bucket.to_owned())
        .max_keys(10) // In this example, go 10 at a time.
        .into_paginator()
        .send();

    while let Some(result) = response.next().await {
        match result {
            Ok(output) => {
                for object in output.contents() {
                    println!(" - {}", object.key().unwrap_or("Unknown"));
                }
            }
            Err(err) => {
                eprintln!("{err:?}")
            }
        }
    }

    Ok(())
}

```

```
/// Given a bucket, remove all objects in the bucket, and then ensure no objects
/// remain in the bucket.
pub async fn clear_bucket(
    client: &aws_sdk_s3::Client,
    bucket_name: &str,
) -> Result<Vec<String>, S3ExampleError> {
    let objects = client.list_objects_v2().bucket(bucket_name).send().await?;

    // delete_objects no longer needs to be mutable.
    let objects_to_delete: Vec<String> = objects
        .contents()
        .iter()
        .filter_map(|obj| obj.key())
        .map(String::from)
        .collect();

    if objects_to_delete.is_empty() {
        return Ok(vec![]);
    }

    let return_keys = objects_to_delete.clone();

    delete_objects(client, bucket_name, objects_to_delete).await?;

    let objects = client.list_objects_v2().bucket(bucket_name).send().await?;

    eprintln!("{objects:?}");

    match objects.key_count {
        Some(0) => Ok(return_keys),
        _ => Err(S3ExampleError::new(
            "There were still objects left in the bucket.",
        )),
    }
}

pub async fn delete_bucket(
    client: &aws_sdk_s3::Client,
    bucket_name: &str,
) -> Result<(), S3ExampleError> {
    let resp = client.delete_bucket().bucket(bucket_name).send().await;
    match resp {
        Ok(_) => Ok(()),
        Err(err) => {
```

```
        if err
            .as_service_error()
            .and_then(aws_sdk_s3::error::ProvideErrorMetadata::code)
            == Some("NoSuchBucket")
        {
            Ok(())
        } else {
            Err(S3ExampleError::from(err))
        }
    }
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der API-Referenz zu AWS - SDK für Rust.
 - [CopyObject](#)
 - [CreateBucket](#)
 - [DeleteBucket](#)
 - [DeleteObjects](#)
 - [GetObject](#)
 - [ListObjectsV2](#)
 - [PutObject](#)

Aktionen

CompleteMultipartUpload

Das folgende Codebeispiel zeigt die Verwendung `CompleteMultipartUpload`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

// 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(),
);
}
```

- Einzelheiten zur API finden Sie [CompleteMultipartUpload](#) in der API-Referenz zum AWS SDK für Rust.

CopyObject

Das folgende Codebeispiel zeigt, wie man es benutzt `CopyObject`.

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/// 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(())
}

```

- Einzelheiten zur API finden Sie [CopyObject](#) in der API-Referenz zum AWS SDK für Rust.

CreateBucket

Das folgende Codebeispiel zeigt, wie man es benutzt `CreateBucket`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

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))
        }
    })
}


```

- Einzelheiten zur API finden Sie [CreateBucket](#) in der API-Referenz zum AWS SDK für Rust.

CreateMultipartUpload

Das folgende Codebeispiel zeigt, wie man es benutzt `CreateMultipartUpload`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// 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?;

```

- Einzelheiten zur API finden Sie [CreateMultipartUpload](#) in der API-Referenz zum AWS SDK für Rust.

DeleteBucket

Das folgende Codebeispiel zeigt, wie man es benutzt `DeleteBucket`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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))
            }
        }
    }
}
```

- Einzelheiten zur API finden Sie [DeleteBucket](#) in der API-Referenz zum AWS SDK für Rust.

DeleteObject

Das folgende Codebeispiel zeigt, wie man es benutzt `DeleteObject`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

- Einzelheiten zur API finden Sie [DeleteObject](#) in der API-Referenz zum AWS SDK für Rust.

DeleteObjects

Das folgende Codebeispiel zeigt, wie man es benutztDeleteObjects.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

/// 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(())
}

```

- Einzelheiten zur API finden Sie [DeleteObjects](#) in der API-Referenz zum AWS SDK für Rust.

GetBucketLocation

Das folgende Codebeispiel zeigt, wie man es benutzt `GetBucketLocation`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}

```

- Einzelheiten zur API finden Sie [GetBucketLocation](#) in der API-Referenz zum AWS SDK für Rust.

GetObject

Das folgende Codebeispiel zeigt, wie man es benutzt `GetObject`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

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)
}

```

- Einzelheiten zur API finden Sie [GetObject](#) in der API-Referenz zum AWS SDK für Rust.

ListBuckets

Das folgende Codebeispiel zeigt, wie man es benutzt `ListBuckets`.

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

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(())
}


```

- Einzelheiten zur API finden Sie [ListBuckets](#) in der API-Referenz zum AWS SDK für Rust.

ListObjectVersions

Das folgende Codebeispiel zeigt, wie man es benutzt `ListObjectVersions`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [ListObjectVersions](#) in der API-Referenz zum AWS SDK für Rust.

ListObjectsV2

Das folgende Codebeispiel zeigt, wie man es benutzt `ListObjectsV2`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}

```

- Einzelheiten zur API finden Sie unter [ListObjectsV2](#) in der API-Referenz zum AWS SDK für Rust.

PutObject

Das folgende Codebeispiel zeigt die Verwendung `PutObject`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

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)
}

```

- Einzelheiten zur API finden Sie [PutObject](#) in der API-Referenz zum AWS SDK für Rust.

UploadPart

Das folgende Codebeispiel zeigt, wie man es benutzt `UploadPart`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

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?;
```

- Einzelheiten zur API finden Sie [UploadPart](#) in der API-Referenz zum AWS SDK für Rust.

Szenarien

Text in Sprache und zurück in Text konvertieren

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Verwenden Sie Amazon Polly, um eine Nur-Text-Eingabedatei (UTF-8) in eine Audiodatei zu synthetisieren.
- Laden Sie die Audiodatei in einen Amazon-S3-Bucket hoch.
- Konvertieren Sie die Audiodatei mit Amazon Transcribe in Text.
- Zeigen Sie den Text an.

SDK für Rust

Verwenden Sie Amazon Polly, um eine Klartext-Eingabedatei (UTF-8) in eine Audiodatei zu synthetisieren, die Audiodatei in einen Amazon-S3-Bucket hochzuladen, diese Audiodatei mit Amazon Transcribe in Text zu konvertieren und den Text anzuzeigen.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- Amazon Polly
- Amazon S3
- Amazon Transcribe

Eine vorsignierte URL erstellen

Das folgende Codebeispiel zeigt, wie Sie eine vorsignierte URL für Amazon S3 erstellen und ein Objekt hochladen.

SDK für Rust

Note

Es gibt noch mehr dazu. GitHub Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen Sie Presigning-Anfragen für GET S3-Objekte.

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

Erstellen Sie Vorsignierungsanforderungen für PUT S3-Objekte.

```
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())
}
```

Erstellen einer Serverless-Anwendung zur Verwaltung von Fotos

Das folgende Codebeispiel zeigt, wie eine Serverless-Anwendung erstellt wird, mit der Benutzer Fotos mithilfe von Labels erstellen können.

SDK für Rust

Zeigt, wie eine Anwendung zur Verwaltung von Fotobeständen entwickelt wird, die mithilfe von Amazon Rekognition Labels in Bildern erkennt und sie für einen späteren Abruf speichert.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

Einen tiefen Einblick in den Ursprung dieses Beispiels finden Sie im Beitrag in der [AWS - Community](#).

In diesem Beispiel verwendete Dienste

- API Gateway

- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Gesichter in einem Bild erkennen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Speichern Sie ein Bild in einem Amazon-S3-Bucket.
- Verwenden Sie Amazon Rekognition, um Gesichtsdetails wie Altersgruppe, Geschlecht und Emotionen (z B. Lächeln) zu erkennen.
- Zeigen Sie diese Details an.

SDK für Rust

Speichern Sie das Bild in einem Amazon-S3-Bucket mit einem uploads-Präfix, verwenden Sie Amazon Rekognition, um Gesichtsdetails wie Altersgruppe, Geschlecht und Emotionen (Lächeln usw.) zu erkennen, und zeigen Sie diese Details an.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- Amazon Rekognition
- Amazon S3

Abrufen eines Objekts aus einem Bucket abrufen, wenn es geändert wurde

Das folgende Codebeispiel zeigt, wie Sie Daten aus einem Objekt in einem S3 Bucket lesen, jedoch nur, wenn dieser Bucket seit dem letzten Abruf nicht geändert wurde.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
use aws_sdk_s3::{
    error::SdkError,
    primitives::{ByteStream, DateTime, DateTimeFormat},
    Client,
};
use s3_code_examples::error::S3ExampleError;
use tracing::{error, warn};

const KEY: &str = "key";
const BODY: &str = "Hello, world!";

/// Demonstrate how `if-modified-since` reports that matching objects haven't
/// changed.
///
/// # Steps
/// - Create a bucket.
/// - Put an object in the bucket.
/// - Get the bucket headers.
/// - Get the bucket headers again but only if modified.
/// - Delete the bucket.
#[tokio::main]
async fn main() -> Result<(), S3ExampleError> {
    tracing_subscriber::fmt::init();

    // Get a new UUID to use when creating a unique bucket name.
    let uuid = uuid::Uuid::new_v4();

    // Load the AWS configuration from the environment.
    let client = Client::new(&aws_config::load_from_env().await);

    // Generate a unique bucket name using the previously generated UUID.
    // Then create a new bucket with that name.
    let bucket_name = format!("if-modified-since-{{uuid}}");
    client
```

```
.create_bucket()
.bucket(bucket_name.clone())
.send()
.await?;

// Create a new object in the bucket whose name is `KEY` and whose
// contents are `BODY`.
let put_object_output = client
    .put_object()
    .bucket(bucket_name.as_str())
    .key(KEY)
    .body(ByteStream::from_static(BODY.as_bytes()))
    .send()
    .await;

// If the `PutObject` succeeded, get the eTag string from it. Otherwise,
// report an error and return an empty string.
let e_tag_1 = match put_object_output {
    Ok(put_object) => put_object.e_tag.unwrap(),
    Err(err) => {
        error!("{err:?}");
        String::new()
    }
};

// Request the object's headers.
let head_object_output = client
    .head_object()
    .bucket(bucket_name.as_str())
    .key(KEY)
    .send()
    .await;

// If the `HeadObject` request succeeded, create a tuple containing the
// values of the headers `last-modified` and `etag`. If the request
// failed, return the error in a tuple instead.
let (last_modified, e_tag_2) = match head_object_output {
    Ok(head_object) => (
        Ok(head_object.last_modified().cloned().unwrap()),
        head_object.e_tag.unwrap(),
    ),
    Err(err) => (Err(err), String::new()),
};
```

```
warn!("last modified: {last_modified:?}");
assert_eq!(
    e_tag_1, e_tag_2,
    "PutObject and first GetObject had differing eTags"
);

println!("First value of last_modified: {last_modified:?}");
println!("First tag: {}\n", e_tag_1);

// Send a second `HeadObject` request. This time, the `if_modified_since`
// option is specified, giving the `last_modified` value returned by the
// first call to `HeadObject`.
//
// Since the object hasn't been changed, and there are no other objects in
// the bucket, there should be no matching objects.

let head_object_output = client
    .head_object()
    .bucket(bucket_name.as_str())
    .key(KEY)
    .if_modified_since(last_modified.unwrap())
    .send()
    .await;

// If the `HeadObject` request succeeded, the result is a tuple containing
// the `last_modified` and `e_tag_1` properties. This is not the expected
// result.
//
// The expected result of the second call to `HeadObject` is an
// `SdkError::ServiceError` containing the HTTP error response. If that's
// the case and the HTTP status is 304 (not modified), the output is a
// tuple containing the values of the HTTP `last-modified` and `etag`
// headers.
//
// If any other HTTP error occurred, the error is returned as an
// `SdkError::ServiceError`.

let (last_modified, e_tag_2) = match head_object_output {
    Ok(head_object) => (
        Ok(head_object.last_modified().cloned().unwrap()),
        head_object.e_tag.unwrap(),
    ),
    Err(err) => match err {
        SdkError::ServiceError(err) => {
```

```
    // Get the raw HTTP response. If its status is 304, the
    // object has not changed. This is the expected code path.
    let http = err.raw();
    match http.status().as_u16() {
        // If the HTTP status is 304: Not Modified, return a
        // tuple containing the values of the HTTP
        // `last-modified` and `etag` headers.
        304 => (
            Ok(DateTime::from_str(
                http.headers().get("last-modified").unwrap(),
                DateTimeFormat::HttpDate,
            )
            .unwrap()),
            http.headers().get("etag").map(|t| t.into()).unwrap(),
        ),
        // Any other HTTP status code is returned as an
        // `SdkError::ServiceError`.
        _ => (Err(SdkError::ServiceError(err)), String::new()),
    }
}
// Any other kind of error is returned in a tuple containing the
// error and an empty string.
_ => (Err(err), String::new()),
},
};

warn!("last modified: {last_modified:?}");
assert_eq!(
    e_tag_1, e_tag_2,
    "PutObject and second HeadObject had different eTags"
);

println!("Second value of last modified: {last_modified:?}");
println!("Second tag: {}", e_tag_2);

// Clean up by deleting the object and the bucket.
client
    .delete_object()
    .bucket(bucket_name.as_str())
    .key(KEY)
    .send()
    .await?;

client
```

```
        .delete_bucket()
        .bucket(bucket_name.as_str())
        .send()
        .await?;

    Ok(())
}
```

- Einzelheiten zur API finden Sie [GetObject](#) in der API-Referenz zum AWS SDK für Rust.

EXIF- und andere Bildinformationen speichern

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Rufen Sie EXIF-Informationen aus einer JPG-, JPEG- oder PNG-Datei ab.
- Laden Sie die Bilddatei in einen Amazon-S3-Bucket hoch.
- Verwenden Sie Amazon Rekognition, um die drei wichtigsten Attribute (Labels) in der Datei zu identifizieren.
- Fügen Sie die EXIF- und Label-Informationen einer Amazon-DynamoDB-Tabelle in der Region hinzu.

SDK für Rust

Rufen Sie EXIF-Informationen aus einer JPG-, JPEG- oder PNG-Datei ab, laden Sie die Bilddatei in einen Amazon-S3-Bucket hoch und identifizieren Sie mit Amazon Rekognition die drei wichtigsten Attribute (Labels in Amazon Rekognition) in der Datei. Fügen Sie die EXIF- und Labelinformationen dann einer Amazon-DynamoDB-Tabelle in der Region hinzu.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- DynamoDB
- Amazon Rekognition
- Amazon S3

Modul- und Integrationstest mit einem SDK

Das folgende Codebeispiel zeigt Beispiele für bewährte Techniken beim Schreiben von Unit- und Integrationstests mithilfe eines AWS SDK.

SDK für Rust

Note

Es gibt noch mehr dazu. GitHub Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Cargo.toml für Testbeispiele

```
[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"
```

Beispiel für Modultests mit automock und einem 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 {
    #[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);
}
}

```

Beispiel für Integrationstests mit StaticReplayClient.

```

use aws_sdk_s3 as s3;

#[allow(dead_code)]
pub async fn determine_prefix_file_size(
    // Now we take a reference to our trait object instead of the S3 client
    // s3_list: ListObjectsService,
    s3: s3::Client,
    bucket: &str,
    prefix: &str,
) -> Result<usize, s3::Error> {
    let mut next_token: Option<String> = None;
    let mut total_size_bytes = 0;
    loop {
        let result = s3
            .list_objects_v2()
            .prefix(prefix)
            .bucket(bucket)
            .set_continuation_token(next_token.take())
            .send()
            .await?;

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

```

```

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

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

#[cfg(test)]
mod test {
    use super::*;
    use aws_config::BehaviorVersion;
    use aws_sdk_s3 as s3;
    use aws_smithy_runtime::client::http::test_util::{ReplayEvent,
StaticReplayClient};
    use aws_smithy_types::body::SdkBody;

    #[tokio::test]
    async fn test_single_page() {
        let page_1 = ReplayEvent::new(
            http::Request::builder()
                .method("GET")
                .uri("https://test-bucket.s3.us-east-1.amazonaws.com/?list-
type=2&prefix=test-prefix")
                .body(SdkBody::empty())
                .unwrap(),
            http::Response::builder()
                .status(200)
                .body(SdkBody::from(include_str!("../testing/response_1.xml")))
                .unwrap(),
        );
    }
}

```

```
let replay_client = StaticReplayClient::new(vec![page_1]);
let client: s3::Client = s3::Client::from_conf(
    s3::Config::builder()
        .behavior_version(BehaviorVersion::latest())
        .credentials_provider(make_s3_test_credentials())
        .region(s3::config::Region::new("us-east-1"))
        .http_client(replay_client.clone())
        .build(),
);

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

// Verify we got the correct total size back
assert_eq!(7, size);
replay_client.assert_requests_match(&[]);
}

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

```
        .body(SdkBody::from(include_str!("./testing/
response_multi_2.xml")))
        .unwrap(),
    );
    let replay_client = StaticReplayClient::new(vec![page_1, page_2]);
    let client: s3::Client = s3::Client::from_conf(
        s3::Config::builder()
            .behavior_version(BehaviorVersion::latest())
            .credentials_provider(make_s3_test_credentials())
            .region(s3::config::Region::new("us-east-1"))
            .http_client(replay_client.clone())
            .build(),
    );

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

    assert_eq!(19, size);

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

Hoch- oder Herunterladen großer Dateien

Das folgende Codebeispiel zeigt, wie große Dateien zu und von Amazon S3 hoch- oder heruntergeladen werden.

Weitere Informationen finden Sie unter [Hochladen eines Objekts mit Multipart-Upload](#).

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
use std::fs::File;
use std::io::prelude::*;
use std::path::Path;

use aws_config::meta::region::RegionProviderChain;
use aws_sdk_s3::error::DisplayErrorContext;
use aws_sdk_s3::operation::{
    create_multipart_upload::CreateMultipartUploadOutput,
    get_object::GetObjectOutput,
};
use aws_sdk_s3::types::{CompletedMultipartUpload, CompletedPart};
use aws_sdk_s3::{config::Region, Client as S3Client};
use aws_smithy_types::byte_stream::{ByteStream, Length};
use rand::distributions::Alphanumeric;
use rand::{thread_rng, Rng};
use s3_code_examples::error::S3ExampleError;
use std::process;
use uuid::Uuid;

//In bytes, minimum chunk size of 5MB. Increase CHUNK_SIZE to send larger chunks.
const CHUNK_SIZE: u64 = 1024 * 1024 * 5;
const MAX_CHUNKS: u64 = 10000;

#[tokio::main]
pub async fn main() {
    if let Err(err) = run_example().await {
        eprintln!("Error: {}", DisplayErrorContext(err));
        process::exit(1);
    }
}

async fn run_example() -> Result<(), S3ExampleError> {
    let shared_config = aws_config::load_from_env().await;
    let client = S3Client::new(&shared_config);

    let bucket_name = format!("amzn-s3-demo-bucket-{}", Uuid::new_v4());
    let region_provider = RegionProviderChain::first_try(Region::new("us-west-2"));
    let region = region_provider.region().await.unwrap();
    s3_code_examples::create_bucket(&client, &bucket_name, &region).await?;

    let key = "sample.txt".to_string();
    // Create a multipart upload. Use UploadPart and CompleteMultipartUpload to
```

```
// upload the file.
let multipart_upload_res: CreateMultipartUploadOutput = client
    .create_multipart_upload()
    .bucket(&bucket_name)
    .key(&key)
    .send()
    .await?;

let upload_id = multipart_upload_res.upload_id().ok_or(S3ExampleError::new(
    "Missing upload_id after CreateMultipartUpload",
))?;

//Create a file of random characters for the upload.
let mut file = File::create(&key).expect("Could not create sample file.");
// Loop until the file is 5 chunks.
while file.metadata().unwrap().len() <= CHUNK_SIZE * 4 {
    let rand_string: String = thread_rng()
        .sample_iter(&Alphanumeric)
        .take(256)
        .map(char::from)
        .collect();
    let return_string: String = "\n".to_string();
    file.write_all(rand_string.as_ref())
        .expect("Error writing to file.");
    file.write_all(return_string.as_ref())
        .expect("Error writing to file.");
}

let path = Path::new(&key);
let file_size = tokio::fs::metadata(path)
    .await
    .expect("it exists I swear")
    .len();

let mut chunk_count = (file_size / CHUNK_SIZE) + 1;
let mut size_of_last_chunk = file_size % CHUNK_SIZE;
if size_of_last_chunk == 0 {
    size_of_last_chunk = CHUNK_SIZE;
    chunk_count -= 1;
}

if file_size == 0 {
    return Err(S3ExampleError::new("Bad file size."));
}
```

```
if chunk_count > MAX_CHUNKS {
    return Err(S3ExampleError::new(
        "Too many chunks! Try increasing your chunk size.",
    ));
}

let mut upload_parts: Vec<aws_sdk_s3::types::CompletedPart> = Vec::new();

for chunk_index in 0..chunk_count {
    let this_chunk = if chunk_count - 1 == chunk_index {
        size_of_last_chunk
    } else {
        CHUNK_SIZE
    };
    let stream = ByteStream::read_from()
        .path(path)
        .offset(chunk_index * CHUNK_SIZE)
        .length(Length::Exact(this_chunk))
        .build()
        .await
        .unwrap();

    // Chunk index needs to start at 0, but part numbers start at 1.
    let part_number = (chunk_index as i32) + 1;
    let upload_part_res = client
        .upload_part()
        .key(&key)
        .bucket(&bucket_name)
        .upload_id(upload_id)
        .body(stream)
        .part_number(part_number)
        .send()
        .await?;

    upload_parts.push(
        CompletedPart::builder()
            .e_tag(upload_part_res.e_tag.unwrap_or_default())
            .part_number(part_number)
            .build(),
    );
}

// upload_parts: Vec<aws_sdk_s3::types::CompletedPart>
```



```
    let completed_multipart_upload: CompletedMultipartUpload =
CompletedMultipartUpload::builder()
    .set_parts(Some(upload_parts))
    .build();

    let _complete_multipart_upload_res = client
        .complete_multipart_upload()
        .bucket(&bucket_name)
        .key(&key)
        .multipart_upload(completed_multipart_upload)
        .upload_id(upload_id)
        .send()
        .await?;

    let data: GetObjectOutput =
        s3_code_examples::download_object(&client, &bucket_name, &key).await?;
    let data_length: u64 = data
        .content_length()
        .unwrap_or_default()
        .try_into()
        .unwrap();
    if file.metadata().unwrap().len() == data_length {
        println!("Data lengths match.");
    } else {
        println!("The data was not the same size!");
    }

    s3_code_examples::clear_bucket(&client, &bucket_name)
        .await
        .expect("Error emptying bucket.");
    s3_code_examples::delete_bucket(&client, &bucket_name)
        .await
        .expect("Error deleting bucket.");

    Ok(())
}
```

Serverless-Beispiele

Aufrufen einer Lambda-Funktion über einen Amazon-S3-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch das Hochladen eines Objekts in einen S3-Bucket ausgelöst wird. Die Funktion ruft den Namen des S3-Buckets sowie den Objektschlüssel aus dem Ereignisparameter ab und ruft die Amazon-S3-API auf, um den Inhaltstyp des Objekts abzurufen und zu protokollieren.

SDK für Rust

Note

Es gibt noch mehr dazu. GitHub Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Nutzen eines S3-Ereignisses mit Lambda unter Verwendung von 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 KI-Beispiele, die SDK für Rust verwenden

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe des AWS SDK für Rust mit SageMaker KI Aktionen ausführen und gängige Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

ListNotebookInstances

Das folgende Codebeispiel zeigt die Verwendung `ListNotebookInstances`.

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [ListNotebookInstances](#) in der API-Referenz zum AWS SDK für Rust.

ListTrainingJobs

Das folgende Codebeispiel zeigt, wie man es benutzt `ListTrainingJobs`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [ListTrainingJobs](#) in der API-Referenz zum AWS SDK für Rust.

Secrets Manager Manager-Beispiele mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe des AWS SDK für Rust mit Secrets Manager Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

GetSecretValue

Das folgende Codebeispiel zeigt die Verwendung `GetSecretValue`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [GetSecretValue](#) in der API-Referenz zum AWS SDK für Rust.

Amazon SES API v2-Beispiele mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe des AWS SDK für Rust mit Amazon SES API v2 Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Szenarien sind Code-Beispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Services aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Themen


- [Aktionen](#)
- [Szenarien](#)

Aktionen

CreateContact

Das folgende Codebeispiel zeigt die Verwendung `CreateContact`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [CreateContact](#) in der API-Referenz zum AWS SDK für Rust.

CreateContactList

Das folgende Codebeispiel zeigt, wie man es benutzt `CreateContactList`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}

```

- Einzelheiten zur API finden Sie [CreateContactList](#) in der API-Referenz zum AWS SDK für Rust.

CreateEmailIdentity

Das folgende Codebeispiel zeigt, wie man es benutzt `CreateEmailIdentity`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

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) ),
    },
}

```

```
}
```

- Einzelheiten zur API finden Sie [CreateEmailIdentity](#) in der API-Referenz zum AWS SDK für Rust.

CreateEmailTemplate

Das folgende Codebeispiel zeigt, wie man es benutzt `CreateEmailTemplate`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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)),
},
}

```

- Einzelheiten zur API finden Sie [CreateEmailTemplate](#) in der API-Referenz zum AWS SDK für Rust.

DeleteContactList

Das folgende Codebeispiel zeigt, wie man es benutzt `DeleteContactList`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

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}")),
}

```

- Einzelheiten zur API finden Sie [DeleteContactList](#) in der API-Referenz zum AWS SDK für Rust.

DeleteEmailIdentity

Das folgende Codebeispiel zeigt, wie man es benutzt `DeleteEmailIdentity`.

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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));
    }
}
```

- Einzelheiten zur API finden Sie [DeleteEmailIdentity](#) in der API-Referenz zum AWS SDK für Rust.

DeleteEmailTemplate

Das folgende Codebeispiel zeigt, wie man es benutzt `DeleteEmailTemplate`.

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

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

- Einzelheiten zur API finden Sie [DeleteEmailTemplate](#) in der API-Referenz zum AWS SDK für Rust.

GetEmailIdentity

Das folgende Codebeispiel zeigt, wie man es benutzt `GetEmailIdentity`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Bestimmt, ob eine E-Mail-Adresse überprüft wurde.

```
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(())
}
```

- Einzelheiten zur API finden Sie [GetEmailIdentity](#) in der API-Referenz zum AWS SDK für Rust.

ListContactLists

Das folgende Codebeispiel zeigt, wie man es benutzt `ListContactLists`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [ListContactLists](#) in der API-Referenz zum AWS SDK für Rust.

ListContacts

Das folgende Codebeispiel zeigt, wie man es benutzt `ListContacts`.

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [ListContacts](#) in der API-Referenz zum AWS SDK für Rust.

SendEmail

Das folgende Codebeispiel zeigt, wie man es benutzt `SendEmail`.

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Sendet eine Nachricht an alle Mitglieder der Kontaktliste.

```
async fn send_message(
    client: &Client,
    list: &str,
    from: &str,
    subject: &str,
    message: &str,
) -> Result<(), Error> {
    // Get list of email addresses from contact list.
    let resp = client
        .list_contacts()
        .contact_list_name(list)
        .send()
        .await?;

    let contacts = resp.contacts();

    let cs: Vec<String> = contacts
        .iter()
        .map(|i| i.email_address().unwrap_or_default().to_string())
        .collect();

    let mut dest: Destination = Destination::builder().build();
    dest.to_addresses = Some(cs);
    let subject_content = Content::builder()
        .data(subject)
        .charset("UTF-8")
        .build()
        .expect("building Content");
    let body_content = Content::builder()
        .data(message)
        .charset("UTF-8")
        .build()
        .expect("building Content");
    let body = Body::builder().text(body_content).build();

    let msg = Message::builder()
        .subject(subject_content)
        .body(body)
        .build();

    let email_content = EmailContent::builder().simple(msg).build();

    client
```



```

        .send_email()
        .from_email_address(from)
        .destination(dest)
        .content(email_content)
        .send()
        .await?;

println!("Email sent to list");

Ok(())
}

```

Sendet mithilfe einer Vorlage eine Nachricht an alle Mitglieder der Kontaktliste.

```

    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)),
    }

```

- Einzelheiten zur API finden Sie [SendEmail](#) in der API-Referenz zum AWS SDK für Rust.

Szenarien

Newsletter-Szenario

Das folgende Codebeispiel zeigt, wie das Amazon SES API v2-Newsletter-Szenario ausgeführt wird.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

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}"));  
}  
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der API-Referenz zu AWS - SDK für Rust.
 - [CreateContact](#)
 - [CreateContactList](#)
 - [CreateEmailIdentity](#)
 - [CreateEmailTemplate](#)
 - [DeleteContactList](#)
 - [DeleteEmailIdentity](#)
 - [DeleteEmailTemplate](#)
 - [ListContacts](#)
 - [SendEmail. einfach](#)
 - [SendEmail. Vorlage](#)

Amazon SNS SNS-Beispiele mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe des AWS SDK für Rust mit Amazon SNS Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Szenarien sind Code-Beispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Services aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Themen

- [Aktionen](#)

- [Szenarien](#)
- [Serverless-Beispiele](#)

Aktionen

CreateTopic

Das folgende Codebeispiel zeigt die Verwendung `CreateTopic`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [CreateTopic](#) in der API-Referenz zum AWS SDK für Rust.

ListTopics

Das folgende Codebeispiel zeigt, wie man es benutzt `ListTopics`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [ListTopics](#) in der API-Referenz zum AWS SDK für Rust.

Publish

Das folgende Codebeispiel zeigt, wie man es benutzt `Publish`.

SDK für Rust

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Details zu API finden Sie unter [Veröffentlichen](#) in der Referenz für das AWS -SDK für Rust-API.

Subscribe

Das folgende Codebeispiel zeigt die Verwendung `Subscribe`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Abonnieren Sie eine E-Mail-Adresse für ein Thema.

```
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(())
}
```

- Details zu API finden Sie unter [Abonnieren](#) in der AWS -SDK-für-Rust-API-Referenz.

Szenarien

Erstellen einer Serverless-Anwendung zur Verwaltung von Fotos

Das folgende Codebeispiel zeigt, wie eine Serverless-Anwendung erstellt wird, mit der Benutzer Fotos mithilfe von Labels erstellen können.

SDK für Rust

Zeigt, wie eine Anwendung zur Verwaltung von Fotobeständen entwickelt wird, die mithilfe von Amazon Rekognition Labels in Bildern erkennt und sie für einen späteren Abruf speichert.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

Einen tiefen Einblick in den Ursprung dieses Beispiels finden Sie im Beitrag in der [AWS - Community](#).

In diesem Beispiel verwendete Dienste

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

Serverless-Beispiele

Eine Lambda-Funktion über einen Amazon-SNS-Trigger aufrufen

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Nachrichten von einem SNS-Thema ausgelöst wird. Die Funktion ruft die Nachrichten aus dem Ereignisparameter ab und protokolliert den Inhalt jeder Nachricht.

SDK für Rust

Note

Es gibt noch mehr dazu. [GitHub](#) Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Nutzen eines SNS-Ereignisses mit Lambda unter Verwendung von 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;

// 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 SQS-Beispiele mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe des AWS SDK für Rust mit Amazon SQS Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Themen

- [Aktionen](#)
- [Serverless-Beispiele](#)

Aktionen

ListQueues

Das folgende Codebeispiel zeigt die Verwendung `ListQueues`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Rufen Sie die erste Amazon SQS SQS-Warteschlange ab, die in der Region aufgeführt ist.

```
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())
}
```

- Einzelheiten zur API finden Sie [ListQueues](#) in der API-Referenz zum AWS SDK für Rust.

ReceiveMessage

Das folgende Codebeispiel zeigt, wie man es benutzt `ReceiveMessage`.

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [ReceiveMessage](#) in der API-Referenz zum AWS SDK für Rust.

SendMessage

Das folgende Codebeispiel zeigt, wie man es benutzt `SendMessage`.

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [SendMessage](#) in der API-Referenz zum AWS SDK für Rust.

Serverless-Beispiele

Aufrufen einer Lambda-Funktion über einen Amazon-SQS-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Nachrichten aus einer SQS-Warteschlange ausgelöst wird. Die Funktion ruft die Nachrichten aus dem Ereignisparameter ab und protokolliert den Inhalt jeder Nachricht.

SDK für Rust

Note

Es gibt noch mehr dazu. [GitHub](#) Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Nutzen eines SQS-Ereignisses mit Lambda unter Verwendung von 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
}
```

Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem Amazon-SQS-Auslöser

Das folgende Codebeispiel zeigt, wie eine partielle Batch-Antwort für Lambda-Funktionen implementiert wird, die Ereignisse aus einer SQS-Warteschlange empfangen. Die Funktion meldet die Batch-Elementfehler in der Antwort und signalisiert Lambda, diese Nachrichten später erneut zu versuchen.

SDK für Rust

 Note

Es gibt noch mehr dazu. [GitHub](#) Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Melden von Fehlern bei SQS-Batchelementen mit Lambda unter Verwendung von 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_identifer: record.message_id.unwrap(),
            }),
        }
    }

    Ok(SqsBatchResponse {
        batch_item_failures,
    })
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    run(service_fn(function_handler)).await
}
```

AWS STS Beispiele mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie das AWS SDK für Rust mit verwenden AWS STS.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

AssumeRole

Das folgende Codebeispiel zeigt die Verwendung AssumeRole.

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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),
}
}
```

- Einzelheiten zur API finden Sie [AssumeRole](#) in der API-Referenz zum AWS SDK für Rust.

Systems Manager Manager-Beispiele mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe des AWS SDK für Rust mit Systems Manager Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

DescribeParameters

Das folgende Codebeispiel zeigt die Verwendung `DescribeParameters`.

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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(())
}
```

- Einzelheiten zur API finden Sie [DescribeParameters](#) in der API-Referenz zum AWS SDK für Rust.

GetParameter

Das folgende Codebeispiel zeigt, wie man es benutzt `GetParameter`.

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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)
}

```

- Einzelheiten zur API finden Sie [GetParameter](#) in der API-Referenz zum AWS SDK für Rust.

PutParameter

Das folgende Codebeispiel zeigt, wie man es benutzt `PutParameter`.

SDK für Rust

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

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(())
}
```

- Einzelheiten zur API finden Sie [PutParameter](#) in der API-Referenz zum AWS SDK für Rust.

Amazon Transcribe Transcribe-Beispiele mit SDK für Rust

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe des AWS SDK für Rust mit Amazon Transcribe Aktionen ausführen und allgemeine Szenarien implementieren.

Szenarien sind Code-Beispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Services aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Themen

- [Szenarien](#)

Szenarien

Text in Sprache und zurück in Text konvertieren

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Verwenden Sie Amazon Polly, um eine Nur-Text-Eingabedatei (UTF-8) in eine Audiodatei zu synthetisieren.
- Laden Sie die Audiodatei in einen Amazon-S3-Bucket hoch.
- Konvertieren Sie die Audiodatei mit Amazon Transcribe in Text.
- Zeigen Sie den Text an.

SDK für Rust

Verwenden Sie Amazon Polly, um eine Klartext-Eingabedatei (UTF-8) in eine Audiodatei zu synthetisieren, die Audiodatei in einen Amazon-S3-Bucket hochzuladen, diese Audiodatei mit Amazon Transcribe in Text zu konvertieren und den Text anzuzeigen.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- Amazon Polly
- Amazon S3
- Amazon Transcribe

Sicherheit für dieses AWS Produkt oder diese Dienstleistung

Cloud-Sicherheit genießt bei Amazon Web Services (AWS) höchste Priorität. Als AWS -Kunde profitieren Sie von einer Rechenzentrums- und Netzwerkarchitektur, die zur Erfüllung der Anforderungen von Organisationen entwickelt wurden, für die Sicherheit eine kritische Bedeutung hat. Sicherheit ist eine gemeinsame Verantwortung zwischen Ihnen AWS und Ihnen. Im [Modell der übergreifenden Verantwortlichkeit](#) wird Folgendes mit „Sicherheit der Cloud“ bzw. „Sicherheit in der Cloud“ umschrieben:

Sicherheit der Cloud — AWS ist verantwortlich für den Schutz der Infrastruktur, auf der alle in der AWS Cloud angebotenen Dienste ausgeführt werden, und für die Bereitstellung von Diensten, die Sie sicher nutzen können. Unsere Sicherheitsverantwortung hat bei uns höchste Priorität AWS, und die Wirksamkeit unserer Sicherheit wird im Rahmen der [AWS Compliance-Programme](#) regelmäßig von externen Prüfern getestet und verifiziert.

Sicherheit in der Cloud — Ihre Verantwortung richtet sich nach dem von Ihnen genutzten AWS Dienst und anderen Faktoren, wie der Sensibilität Ihrer Daten, den Anforderungen Ihres Unternehmens und den geltenden Gesetzen und Vorschriften.

Dieses AWS Produkt oder dieser Service folgt dem [Modell der gemeinsamen Verantwortung](#) in Bezug auf die spezifischen Amazon Web Services (AWS) -Services, die es unterstützt. Informationen zur AWS Servicesicherheit finden Sie auf der [Seite mit der Dokumentation zur AWS Servicesicherheit](#) und den [AWS Services, für die das AWS Compliance-Programm zur Einhaltung der](#) Vorschriften zuständig ist.

Themen

- [Datenschutz in diesem AWS Produkt oder dieser Dienstleistung](#)
- [Überprüfung der Einhaltung der Vorschriften für dieses AWS Produkt oder diese Dienstleistung](#)
- [Sicherheit der Infrastruktur für dieses AWS Produkt oder diesen Service](#)
- [Erzwingen Sie eine TLS-Mindestversion in AWS SDK for Rust](#)

Datenschutz in diesem AWS Produkt oder dieser Dienstleistung

Das AWS [Modell](#) der gilt für den Datenschutz in diesem AWS Produkt oder dieser Dienstleistung. Wie in diesem Modell beschrieben, AWS ist verantwortlich für den Schutz der globalen Infrastruktur, auf der alle Systeme laufen AWS Cloud. Sie sind dafür verantwortlich, die Kontrolle über Ihre in

dieser Infrastruktur gehosteten Inhalte zu behalten. Sie sind auch für die Sicherheitskonfiguration und die Verwaltungsaufgaben für die von Ihnen verwendeten AWS-Services verantwortlich. Weitere Informationen zum Datenschutz finden Sie unter [Häufig gestellte Fragen zum Datenschutz](#). Informationen zum Datenschutz in Europa finden Sie im Blog-Beitrag [AWS -Modell der geteilten Verantwortung und in der DSGVO](#) im AWS -Sicherheitsblog.

Aus Datenschutzgründen empfehlen wir, dass Sie AWS-Konto Anmeldeinformationen schützen und einzelne Benutzer mit AWS IAM Identity Center oder AWS Identity and Access Management (IAM) einrichten. So erhält jeder Benutzer nur die Berechtigungen, die zum Durchführen seiner Aufgaben erforderlich sind. Außerdem empfehlen wir, die Daten mit folgenden Methoden schützen:

- Verwenden Sie für jedes Konto die Multi-Faktor-Authentifizierung (MFA).
- Verwenden Sie SSL/TLS, um mit Ressourcen zu kommunizieren. AWS Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Richten Sie die API und die Protokollierung von Benutzeraktivitäten mit ein. AWS CloudTrail Informationen zur Verwendung von CloudTrail Pfaden zur Erfassung von AWS Aktivitäten finden Sie unter [Arbeiten mit CloudTrail Pfaden](#) im AWS CloudTrail Benutzerhandbuch.
- Verwenden Sie AWS Verschlüsselungslösungen zusammen mit allen darin enthaltenen Standardsicherheitskontrollen AWS-Services.
- Verwenden Sie erweiterte verwaltete Sicherheitsservices wie Amazon Macie, die dabei helfen, in Amazon S3 gespeicherte persönliche Daten zu erkennen und zu schützen.
- Wenn Sie für den Zugriff AWS über eine Befehlszeilenschnittstelle oder eine API FIPS 140-3-validierte kryptografische Module benötigen, verwenden Sie einen FIPS-Endpunkt. Weitere Informationen über verfügbare FIPS-Endpunkte finden Sie unter [Federal Information Processing Standard \(FIPS\) 140-3](#).

Wir empfehlen dringend, in Freitextfeldern, z. B. im Feld Name, keine vertraulichen oder sensiblen Informationen wie die E-Mail-Adressen Ihrer Kunden einzugeben. Dazu gehört auch, wenn Sie mit diesem AWS Produkt oder Service oder einem anderen AWS-Services über die Konsole, API oder arbeiten. AWS CLI AWS SDKs Alle Daten, die Sie in Tags oder Freitextfelder eingeben, die für Namen verwendet werden, können für Abrechnungs- oder Diagnoseprotokolle verwendet werden. Wenn Sie eine URL für einen externen Server bereitstellen, empfehlen wir dringend, keine Anmeldeinformationen zur Validierung Ihrer Anforderung an den betreffenden Server in die URL einzuschließen.

Überprüfung der Einhaltung der Vorschriften für dieses AWS Produkt oder diese Dienstleistung

Informationen darüber, ob AWS-Service ein [AWS-Services in den Geltungsbereich bestimmter Compliance-Programme fällt](#), finden Sie unter [Umfang nach Compliance-Programm AWS-Services unter](#) . Wählen Sie dort das Compliance-Programm aus, an dem Sie interessiert sind. Allgemeine Informationen finden Sie unter [AWS Compliance-Programme AWS](#) .

Sie können Prüfberichte von Drittanbietern unter herunterladen AWS Artifact. Weitere Informationen finden Sie unter [Berichte herunterladen unter](#) .

Ihre Verantwortung für die Einhaltung der Vorschriften bei der Nutzung AWS-Services hängt von der Vertraulichkeit Ihrer Daten, den Compliance-Zielen Ihres Unternehmens und den geltenden Gesetzen und Vorschriften ab. AWS stellt die folgenden Ressourcen zur Verfügung, die Sie bei der Einhaltung der Vorschriften unterstützen:

- [Compliance und Governance im Bereich Sicherheit](#) – In diesen Anleitungen für die Lösungsimplementierung werden Überlegungen zur Architektur behandelt. Außerdem werden Schritte für die Bereitstellung von Sicherheits- und Compliance-Features beschrieben.
- [Referenz für HIPAA-fähige Dienste](#) — Listet HIPAA-fähige Dienste auf. Nicht alle sind HIPAA-fähig AWS-Services .
- [AWS Compliance-Ressourcen](#) — Diese Sammlung von Arbeitsmappen und Leitfäden gilt möglicherweise für Ihre Branche und Ihren Standort.
- [AWS Leitfäden zur Einhaltung von Vorschriften für Kunden](#) — Verstehen Sie das Modell der gemeinsamen Verantwortung aus dem Blickwinkel der Einhaltung von Vorschriften. In den Leitfäden werden die bewährten Verfahren zur Sicherung zusammengefasst AWS-Services und die Leitlinien den Sicherheitskontrollen in verschiedenen Frameworks (einschließlich des National Institute of Standards and Technology (NIST), des Payment Card Industry Security Standards Council (PCI) und der International Organization for Standardization (ISO)) zugeordnet.
- [Evaluierung von Ressourcen anhand von Regeln](#) im AWS Config Entwicklerhandbuch — Der AWS Config Service bewertet, wie gut Ihre Ressourcenkonfigurationen den internen Praktiken, Branchenrichtlinien und Vorschriften entsprechen.
- [AWS Security Hub](#)— Auf diese AWS-Service Weise erhalten Sie einen umfassenden Überblick über Ihren internen Sicherheitsstatus. AWS Security Hub verwendet Sicherheitskontrollen, um Ihre AWS -Ressourcen zu bewerten und Ihre Einhaltung von Sicherheitsstandards und bewährten

Methoden zu überprüfen. Eine Liste der unterstützten Services und Kontrollen finden Sie in der [Security-Hub-Steuerungsreferenz](#).

- [Amazon GuardDuty](#) — Dies AWS-Service erkennt potenzielle Bedrohungen für Ihre Workloads AWS-Konten, Container und Daten, indem es Ihre Umgebung auf verdächtige und böswillige Aktivitäten überwacht. GuardDuty kann Ihnen helfen, verschiedene Compliance-Anforderungen wie PCI DSS zu erfüllen, indem es die in bestimmten Compliance-Frameworks vorgeschriebenen Anforderungen zur Erkennung von Eindringlingen erfüllt.
- [AWS Audit Manager](#)— Auf diese AWS-Service Weise können Sie Ihre AWS Nutzung kontinuierlich überprüfen, um das Risikomanagement und die Einhaltung von Vorschriften und Industriestandards zu vereinfachen.

Dieses AWS Produkt oder dieser Service folgt dem [Modell der gemeinsamen Verantwortung](#) in Bezug auf die spezifischen Amazon Web Services (AWS) -Services, die es unterstützt. Informationen zur AWS Servicesicherheit finden Sie auf der [Seite mit der Dokumentation zur AWS Servicesicherheit](#) und den [AWS Services, für die das AWS Compliance-Programm zur Einhaltung der](#) Vorschriften zuständig ist.

Sicherheit der Infrastruktur für dieses AWS Produkt oder diesen Service

Dieses AWS Produkt oder dieser Dienst verwendet Managed Services und ist daher durch die AWS globale Netzwerksicherheit geschützt. Informationen zu AWS Sicherheitsdiensten und zum AWS Schutz der Infrastruktur finden Sie unter [AWS Cloud-Sicherheit](#). Informationen zum Entwerfen Ihrer AWS Umgebung unter Verwendung der bewährten Methoden für die Infrastruktursicherheit finden Sie unter [Infrastructure Protection](#) in Security Pillar AWS Well-Architected Framework.

Sie verwenden AWS veröffentlichte API-Aufrufe, um über das Netzwerk auf dieses AWS Produkt oder diesen Service zuzugreifen. Kunden müssen Folgendes unterstützen:

- Transport Layer Security (TLS). Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Verschlüsselungs-Suiten mit Perfect Forward Secrecy (PFS) wie DHE (Ephemeral Diffie-Hellman) oder ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Die meisten modernen Systeme wie Java 7 und höher unterstützen diese Modi.

Außerdem müssen Anforderungen mit einer Zugriffsschlüssel-ID und einem geheimen Zugriffsschlüssel signiert sein, der einem IAM-Prinzipal zugeordnet ist. Alternativ können Sie mit [AWS](#)

[Security Token Service](#) (AWS STS) temporäre Sicherheitsanmeldeinformationen erstellen, um die Anforderungen zu signieren.

Dieses AWS Produkt oder dieser Service folgt dem [Modell der gemeinsamen Verantwortung](#) in Bezug auf die spezifischen Amazon Web Services (AWS) -Services, die es unterstützt. Informationen zur AWS Servicesicherheit finden Sie auf der [Seite mit der Dokumentation zur AWS Servicesicherheit](#) und den [AWS Services, für die das AWS Compliance-Programm zur Einhaltung der](#) Vorschriften zuständig ist.

Erzwingen Sie eine TLS-Mindestversion in AWS SDK for Rust

The AWS SDK for Rust verwendet TLS, um die Sicherheit bei der Kommunikation mit AWS Diensten zu erhöhen. Das SDK erzwingt standardmäßig eine TLS-Mindestversion von 1.2. Standardmäßig handelt das SDK auch die höchste TLS-Version aus, die sowohl für die Client-Anwendung als auch für den Dienst verfügbar ist. Beispielsweise könnte das SDK in der Lage sein, TLS 1.3 auszuhandeln.

Eine bestimmte TLS-Version kann in der Anwendung erzwungen werden, indem der vom SDK verwendete TCP-Connector manuell konfiguriert wird. Um dies zu veranschaulichen, zeigt Ihnen das folgende Beispiel, wie Sie TLS 1.3 erzwingen können.

Note

Einige AWS Dienste unterstützen TLS 1.3 noch nicht, sodass die Durchsetzung dieser Version die SDK-Interoperabilität beeinträchtigen kann. Wir empfehlen, diese Konfiguration mit jedem Dienst vor der Produktionsbereitstellung zu testen.

```
pub async fn connect_via_tls_13() -> Result<(), Error> {
    println!("Attempting to connect to KMS using TLS 1.3: ");

    // Let webpki load the Mozilla root certificates.
    let mut root_store = RootCertStore::empty();
    root_store.add_server_trust_anchors(webpki_roots::TLS_SERVER_ROOTS.0.iter().map(|
ta| {
        rustls::OwnedTrustAnchor::from_subject_spki_name_constraints(
            ta.subject,
            ta.spki,
            ta.name_constraints,
        )
    }
    )
}
```

```
    }));

    // The .with_protocol_versions call is where we set TLS1.3. You can add
    rustls::version::TLS12 or replace them both with rustls::ALL_VERSIONS
    let config = rustls::ClientConfig::builder()
        .with_safe_default_cipher_suites()
        .with_safe_default_kx_groups()
        .with_protocol_versions(&[&rustls::version::TLS13])
        .expect("It looks like your system doesn't support TLS1.3")
        .with_root_certificates(root_store)
        .with_no_client_auth();

    // Finish setup of the rustls connector.
    let rustls_connector = hyper_rustls::HttpsConnectorBuilder::new()
        .with_tls_config(config)
        .https_only()
        .enable_http1()
        .enable_http2()
        .build();

    // See https://github.com/aws-labs/smithy-rs/discussions/3022 for the
    HyperClientBuilder
    let http_client = HyperClientBuilder::new().build(rustls_connector);

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

    let kms_client = aws_sdk_kms::Client::new(&shared_conf);
    let response = kms_client.list_keys().send().await?;

    println!("{:?}", response);

    Ok(())
}
```

Anhang: Vom SDK verwendete Rostkisten

Dieser Anhang enthält erweiterte Informationen zu den Kisten, die von der verwendet werden. AWS SDK for Rust Dazu gehören die verwendeten Smithy-Komponenten, Kisten, die Sie unter bestimmten Baubedingungen möglicherweise benötigen, und weitere Informationen.

Smithy-Kisten

Das AWS SDK for Rust basiert auf [Smithy](#), wie die meisten. AWS SDKs Smithy ist eine Sprache, die verwendet wird, um die vom SDK angebotenen Datentypen und Funktionen zu beschreiben. Diese Modelle werden dann verwendet, um das SDK selbst zu erstellen.

Wenn Sie sich die Versionen des SDK für Rust-Kisten und die seiner Smithy-Abhängigkeiten ansehen, könnte es hilfreich sein zu wissen, dass diese Kisten alle eine [standardmäßige semantische Versionsnummerierung](#) verwenden.

[Weitere detaillierte Informationen zu Smithy Crates for Rust finden Sie unter Smithy Rust Design.](#)

Mit dem SDK für Rust verwendete Kisten

Es gibt eine Reihe von Smithy-Kisten, die von veröffentlicht wurden. AWS Einige davon sind für SDK für Rust-Benutzer relevant, während es sich bei anderen um Implementierungsdetails handelt:

`aws-smithy-async`

Fügen Sie diese Kiste hinzu, wenn Sie Tokio nicht für asynchrone Funktionen verwenden.

`aws-smithy-runtime`

Enthält Bausteine, die von allen benötigt werden. AWS SDKs

`aws-smithy-runtime-api`

Zugrundeliegende Schnittstellen, die vom SDK verwendet werden.

`aws-smithy-types`

Typen, die aus anderen AWS SDKs erneut exportiert wurden. Verwenden Sie dies, wenn Sie mehrere SDKs verwenden.

`aws-smithy-types-convert`

Hilfsfunktionen zum Ein- und Ausziehen `aws-smithy-types`.

Andere Kisten

Die folgenden Kisten gibt es, aber du solltest nichts über sie wissen müssen:

Serverbezogene Kisten, die SDK für Rust-Benutzer nicht benötigen:

- `aws-smithy-http-server`
- `aws-smithy-http-server-python`

Kisten, die under-the-hood Code enthalten, den SDK-Benutzer nicht verwenden müssen:

- `aws-smithy-checksum-callbacks`
- `aws-smithy-eventstream`
- `aws-smithy-http`
- `aws-smithy-protocol-test`
- `aws-smithy-query`
- `aws-smithy-json`
- `aws-smithy-xml`

Kisten, die nicht unterstützt werden und in future verschwinden werden:

- `aws-smithy-client`
- `aws-smithy-http-auth`
- `aws-smithy-http-tower`

Dokumentverlauf

In diesem Thema werden wichtige Änderungen beschrieben, die im Laufe der Geschichte des AWS SDK for Rust Entwicklerhandbuchs vorgenommen wurden.

Änderung	Beschreibung	Datum
<u>Das Inhaltsverzeichnis wird neu organisiert</u>	Neuorganisation des Inhaltsverzeichnisses, um Konfiguration besser von Verwendung zu unterscheiden.	1. Juli 2024
<u>Allgemeine Verfügbarkeit des AWS SDK for Rust</u>	Das Handbuch wurde aktualisiert und enthält nun neue Sicherheitsinformationen, neue und aktualisierte Codebeispiele, neue Details zu Unit-Tests mit Beispielen und weitere neue und aktualisierte Inhalte für die neue Version des SDK für allgemeine Verfügbarkeit.	8. November 2023
<u>Durchsetzung einer TLS-Mindestversion</u>	Es wurden Informationen zur Durchsetzung einer TLS-Version im SDK hinzugefügt.	4. Mai 2022
<u>AWS SDK for Rust Vorschauversion für Entwickler</u>	<u>Vorschauversion für Entwickler</u>	2. Dezember 2021

Die vorliegende Übersetzung wurde maschinell erstellt. Im Falle eines Konflikts oder eines Widerspruchs zwischen dieser übersetzten Fassung und der englischen Fassung (einschließlich infolge von Verzögerungen bei der Übersetzung) ist die englische Fassung maßgeblich.