

Guida per gli sviluppatori

AWS SDK for Rust



AWS SDK for Rust: Guida per gli sviluppatori

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

I marchi e il trade dress di Amazon non possono essere utilizzati in relazione ad alcun prodotto o servizio che non sia di Amazon, in alcun modo che possa causare confusione tra i clienti, né in alcun modo che possa denigrare o screditare Amazon. Tutti gli altri marchi non di proprietà di Amazon sono di proprietà delle rispettive aziende, che possono o meno essere associate, collegate o sponsorizzate da Amazon.

Table of Contents

Qual è il AWS SDK for Rust?	1
Guida introduttiva all'SDK	1
Manutenzione e supporto per le versioni principali dell'SDK	1
Risorse aggiuntive	1
Inizia a usare	3
Autenticazione SDK con AWS	3
Avvia una sessione del portale di AWS accesso	4
Ulteriori informazioni di autenticazione	5
Ciao tutorial	6
Prerequisiti	6
Crea la tua prima app SDK	6
Nozioni fondamentali	8
Prerequisiti	6
Nozioni di base su Rust	9
AWS SDK for Rust crea i fondamentali	10
Configurazione del progetto con cui lavorare Servizi AWS	10
Runtime Tokio	11
Configura	12
Creare un client di servizio	12
Configura un client dall'ambiente	13
Utilizza il modello builder per le impostazioni specifiche del servizio	14
Configurazione avanzata esplicita del client	15
Versioni comportamentali	16
Imposta la versione del comportamento in <code>Cargo.toml</code>	17
Imposta la versione del comportamento nel codice	17
Fornitori di credenziali	17
La catena di fornitori di credenziali	18
Fornitore di credenziali esplicito	20
Memorizzazione nella cache delle identità	20
Regione AWS selezione	21
Regione AWS catena di fornitori	21
Opzioni di configurazione	22
Endpoint del cliente	22
Configurazione personalizzata	23

Esempi	26
HTTP	28
Iper 1.x	28
Intercettori	30
Registrazione Interceptor	31
Osservabilità	32
Registrazione	33
Sovrascrivi la configurazione operativa	36
Tentativi	38
Configurazione predefinita dei nuovi tentativi	38
Numero massimo di tentativi	39
Ritardi e arretramenti	39
Modalità di riprova adattiva	40
Timeout	40
Timeout delle API	41
Protezione del flusso bloccata	42
Usa l'SDK	44
Fai richieste	44
Best practice	45
Riutilizza i client SDK quando possibile	45
Configura i timeout delle API	46
Simultaneità	46
Termini	46
Un semplice esempio	47
Proprietà e mutabilità	48
Altri termini!	49
Riscrivere il nostro esempio per renderlo più efficiente (concorrenza a thread singolo)	50
Riscrivere il nostro esempio per renderlo più efficiente (concorrenza multithread)	52
Eseguire il debug di app multithread	54
Gestione degli errori	54
Errori del servizio	54
Metadati di errore	55
Errore di stampa dettagliato con <code>DisplayErrorContext</code>	56
Creare funzioni Lambda	58
Paginazione	58
Crea predefiniti URLs	60

Nozioni di base sulla preassegnazione	60
Preassegnazione POST e richieste PUT	61
Firmatario autonomo	62
Test unitari	63
Genera automaticamente dei mock usando mockall	64
Replay statico	69
Waiter	73
Esempi di codice	75
API Gateway	76
Operazioni	77
Scenari	78
AWS contributi della comunità	79
API Gateway Management API	79
Operazioni	77
Application Auto Scaling	81
Operazioni	77
Aurora	82
Nozioni di base	84
Operazioni	77
Auto Scaling	230
Nozioni di base	84
Operazioni	77
Runtime di Amazon Bedrock	264
Anthropic Claude	265
Provider di identità Amazon Cognito	280
Operazioni	77
Amazon Cognito Sync	282
Operazioni	77
Firehose	283
Operazioni	77
Amazon DocumentDB	284
Esempi serverless	285
DynamoDB	286
Operazioni	77
Scenari	78
Esempi serverless	285

AWS contributi della comunità	79
Amazon EBS	305
Operazioni	77
Amazon EC2	307
Nozioni di base	84
Operazioni	77
Amazon ECR	369
Operazioni	77
Amazon ECS	371
Operazioni	77
Amazon EKS	374
Operazioni	77
AWS Glue	376
Nozioni di base	84
Operazioni	77
IAM	391
Nozioni di base	84
Operazioni	77
AWS IoT	418
Operazioni	77
Kinesis	420
Operazioni	77
Esempi serverless	285
AWS KMS	428
Operazioni	77
Lambda	437
Nozioni di base	84
Operazioni	77
Scenari	78
Esempi serverless	285
AWS contributi della comunità	79
MediaLive	488
Operazioni	77
MediaPackage	489
Operazioni	77
MSK Amazon	491

Esempi serverless	285
Amazon Polly	493
Operazioni	77
Scenari	78
QLDB	498
Operazioni	77
Amazon RDS	500
Esempi serverless	285
Servizi di dati di Amazon RDS	504
Operazioni	77
Amazon Rekognition	505
Scenari	78
Route 53	507
Operazioni	77
Amazon S3	509
Nozioni di base	84
Operazioni	77
Scenari	78
Esempi serverless	285
SageMaker AI	559
Operazioni	77
Secrets Manager	561
Operazioni	77
API Amazon SES v2	562
Operazioni	77
Scenari	78
Amazon SNS	578
Operazioni	77
Scenari	78
Esempi serverless	285
Amazon SQS	585
Operazioni	77
Esempi serverless	285
AWS STS	590
Operazioni	77
Systems Manager	591

Operazioni	77
Amazon Transcribe	594
Scenari	78
Sicurezza	596
Protezione dei dati	596
Convalida della conformità	597
Sicurezza dell'infrastruttura	599
Applica una versione TLS minima	599
Casse utilizzate dall'SDK	602
casse Smithy	602
Casse utilizzate con l'SDK	602
Altre casse	603
Cronologia dei documenti	604
.....	dcv

Qual è il AWS SDK for Rust?

Rust è un linguaggio di programmazione di sistemi senza un garbage collector incentrato su tre obiettivi: sicurezza, velocità e concorrenza.

AWS SDK for Rust (l'SDK) fornisce APIs a Rust l'interazione con i servizi di infrastruttura Amazon Web Services. Utilizzando l'SDK, puoi creare applicazioni su Amazon S3, Amazon EC2, DynamoDB e altro ancora.

Argomenti

- [Guida introduttiva all'SDK](#)
- [Manutenzione e supporto per le versioni principali dell'SDK](#)
- [Risorse aggiuntive](#)

Guida introduttiva all'SDK

Se utilizzi l'SDK per la prima volta, ti consigliamo di iniziare leggendo. [Inizia con AWS SDK for Rust](#)

Per la configurazione e la configurazione, incluso come creare e configurare i client di servizio a cui effettuare richieste Servizi AWS, consulta. [Configura il AWS SDK for Rust](#)

Per informazioni sull'utilizzo dell'SDK, consulta [Usa il AWS SDK for Rust](#).

Per un elenco completo degli esempi di codice Rust, consulta [Esempi di codice](#).

Manutenzione e supporto per le versioni principali dell'SDK

Per informazioni sulla manutenzione e il supporto per le versioni principali dell'SDK e le relative dipendenze sottostanti, consulta quanto segue nella Guida di [riferimento agli strumenti AWS SDKs e agli strumenti](#):

- [AWS SDKs e politica di manutenzione degli strumenti](#)
- [AWS SDKs e Tools Version Support Matrix](#)

Risorse aggiuntive

Oltre a questa guida, le seguenti sono preziose risorse online per gli sviluppatori SDK:

- [AWS SDKs e Guida di riferimento agli strumenti](#): contiene impostazioni, funzionalità e altri concetti fondamentali comuni a. AWS SDKs
- [Sito web del linguaggio di programmazione Rust](#)
- [AWS SDK for Rust Documentazione di riferimento API](#)
- [AWS Blog sugli strumenti per sviluppatori](#)
- [AWS SDK for Rust codice sorgente](#) su GitHub
- [Il catalogo di esempi di AWS codice](#)

Inizia con AWS SDK for Rust

Scopri come installare, configurare e utilizzare l'SDK per creare un'applicazione Rust per accedere a una AWS risorsa a livello di codice.

Argomenti

- [Autenticazione SDK con AWS](#)
- [Hello tutorial per AWS SDK for Rust](#)
- [Nozioni fondamentali](#)

Autenticazione SDK con AWS

È necessario stabilire in che modo il codice si autentica durante lo sviluppo con AWS. Servizi AWS È possibile configurare l'accesso programmatico alle AWS risorse in diversi modi a seconda dell'ambiente e dell'AWS accesso a disposizione.

Per scegliere il metodo di autenticazione e configurarlo per l'SDK, consulta [Autenticazione e accesso](#) nella Guida di riferimento agli strumenti AWS SDKs e agli strumenti.

Consigliamo ai nuovi utenti che si stanno sviluppando localmente e che non dispongono di un metodo di autenticazione da parte del datore di lavoro di AWS IAM Identity Center configurarlo. Questo metodo include l'installazione di AWS CLI per facilitare la configurazione e per accedere regolarmente al portale di AWS accesso. Se scegli questo metodo, l'ambiente dovrebbe contenere i seguenti elementi dopo aver completato la procedura per l'[autenticazione di IAM Identity Center](#) nella AWS SDKs and Tools Reference Guide:

- Il AWS CLI, che viene utilizzato per avviare una sessione del portale di AWS accesso prima di eseguire l'applicazione.
- Un [AWSconfigfile condiviso](#) con un [default] profilo con un set di valori di configurazione a cui è possibile fare riferimento dall'SDK. Per trovare la posizione di questo file, consulta [Posizione dei file condivisi nella Guida](#) di riferimento agli strumenti AWS SDKs e strumenti.
- Il config file condiviso imposta l'[region](#) impostazione. Questo imposta l'impostazione predefinita Regione AWS utilizzata dall'SDK per AWS le richieste. Questa regione viene utilizzata per le richieste di servizio SDK che non sono specificate con una regione da utilizzare.
- L'SDK utilizza la [configurazione del provider di token SSO](#) del profilo per acquisire le credenziali prima di inviare richieste a. AWS Il `sso_role_name` valore, che è un ruolo IAM connesso a

un set di autorizzazioni IAM Identity Center, consente l'accesso ai dati Servizi AWS utilizzati nell'applicazione.

Il seguente config file di esempio mostra un profilo predefinito impostato con la configurazione del provider di token SSO. L'`sso_session` impostazione del profilo si riferisce alla [sso-sessione](#) denominata. La `sso-session` sezione contiene le impostazioni per avviare una sessione del portale di AWS accesso.

```
[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
```

L'SDK per Rust non richiede l'aggiunta di pacchetti aggiuntivi (come SSO eSSO0IDC) all'applicazione per utilizzare l'autenticazione IAM Identity Center.

Avvia una sessione del portale di AWS accesso

Prima di eseguire un'applicazione che consente l'accesso Servizi AWS, è necessaria una sessione attiva del portale di AWS accesso affinché l'SDK utilizzi l'autenticazione IAM Identity Center per risolvere le credenziali. A seconda della durata della sessione configurata, l'accesso alla fine scadrà e l'SDK riscontrerà un errore di autenticazione. Per accedere al portale di AWS accesso, esegui il seguente comando in AWS CLI

```
$ aws sso login
```

Se hai seguito le istruzioni e disponi di una configurazione predefinita del profilo, non è necessario richiamare il comando con un `--profile` opzione. Se la configurazione del provider di token SSO utilizza un profilo denominato, il comando è `aws sso login --profile named-profile`.

Per verificare facoltativamente se hai già una sessione attiva, esegui il AWS CLI comando seguente.

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

Se la sessione è attiva, la risposta a questo comando riporta l'account IAM Identity Center e il set di autorizzazioni configurati nel `config` file condiviso.

Note

Se hai già una sessione attiva del portale di AWS accesso ed esegui `aws sso login`, non ti verrà richiesto di fornire credenziali.

La procedura di accesso potrebbe richiedere all'utente di consentire l'AWS CLI accesso ai dati. Poiché AWS CLI è basato sull'SDK per Python, i messaggi di autorizzazione potrebbero contenere variazioni del `botocore` nome.

Ulteriori informazioni di autenticazione

Utenti umani, noti anche come identità umane, sono le persone, gli amministratori, gli sviluppatori, gli operatori e i consumatori delle tue applicazioni. Devono avere un'identità per accedere agli AWS ambienti e alle applicazioni dell'utente. Gli utenti umani che fanno parte della tua organizzazione, ovvero tu, lo sviluppatore, sono noti come identità della forza lavoro.

Utilizza credenziali temporanee per l'accesso. AWS Puoi utilizzare un provider di identità per i tuoi utenti umani per fornire l'accesso federato agli AWS account assumendo ruoli che forniscono credenziali temporanee. Per la gestione centralizzata degli accessi, ti consigliamo di utilizzare AWS IAM Identity Center (IAM Identity Center) per gestire l'accesso ai tuoi account e le autorizzazioni all'interno di tali account. Per altre alternative, consulta quanto segue:

- Per ulteriori informazioni sulle best practice, consulta [Best practice per la sicurezza in IAM](#) nella Guida per l'utente di IAM.
- Per creare AWS credenziali a breve termine, consulta [Temporary Security Credentials](#) nella IAM User Guide.
- Per ulteriori informazioni sugli altri provider di credenziali supportati da SDK for Rust, consulta Fornitori di [credenziali standardizzati nella and Tools Reference Guide](#).AWS SDKs

Hello tutorial per AWS SDK for Rust

Prerequisiti

Per poter utilizzare AWS SDK for Rust, è necessario che Rust and Cargo siano installati.

- Installa la toolchain Rust: <https://www.rust-lang.org/tools/install>
- Installa lo cargo-component [strumento](#) eseguendo il comando: `cargo install cargo-component`

Strumenti consigliati:

I seguenti strumenti opzionali possono essere installati nell'IDE per facilitare il completamento del codice e la risoluzione dei problemi.

- L'estensione rust-analyzer, vedi [Rust in Visual Studio Code](#).
- Amazon Q Developer, consulta [Installazione dell'estensione o del plug-in Amazon Q Developer nel tuo IDE](#).

Crea la tua prima app SDK

Questa procedura crea la prima applicazione SDK per Rust che elenca le tabelle DynamoDB.

1. In una finestra del terminale o della console, accedete alla posizione del computer in cui desiderate creare l'app.
2. Esegui il seguente comando per creare una `hello_world` directory e popolarla con uno scheletro progetto Rust:

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

3. Naviga nella `hello_world` directory e usa il seguente comando per aggiungere le dipendenze richieste all'app:

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

Queste dipendenze includono le casse SDK che forniscono funzionalità di configurazione e supporto per DynamoDB, inclusa la [tokiocassa, utilizzata per implementare operazioni di I/O asincrone](#).

Note

A meno che non si utilizzi una funzionalità come Tokio non fornirà un runtime asincrono. `tokio/full` L'SDK per Rust richiede un runtime asincrono.

4. Aggiorna `main.rs` nella `src` directory per contenere il codice seguente.

```
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

Questo esempio visualizza solo la prima pagina dei risultati. Scopri come gestire più pagine di risultati. [the section called "Paginazione"](#)

5. Esegui il programma:

```
$ cargo run
```

Dovresti vedere un elenco dei nomi delle tue tabelle.

Nozioni fondamentali

Prerequisiti

Per poter utilizzare AWS SDK for Rust, è necessario che Rust and Cargo siano installati.

- Installa la toolchain Rust: <https://www.rust-lang.org/tools/install>
- Installa lo cargo-component [strumento](#) eseguendo il comando: `cargo install cargo-component`

Strumenti consigliati:

I seguenti strumenti opzionali possono essere installati nell'IDE per facilitare il completamento del codice e la risoluzione dei problemi.

- L'estensione rust-analyzer, vedi [Rust in Visual Studio Code](#).
- Amazon Q Developer, consulta [Installazione dell'estensione o del plug-in Amazon Q Developer nel tuo IDE](#).

Nozioni di base su Rust

Di seguito sono riportate alcune nozioni di base del linguaggio di programmazione Rust che sarebbe utile conoscere. Tutti i riferimenti per ulteriori informazioni provengono da [The Rust Programming Language](#).

- `Cargo.toml` è il file di configurazione standard del progetto Rust, contiene le dipendenze e alcuni metadati sul progetto. I file sorgente di Rust hanno un' `.rs` estensione di file. Vedi [Hello, Cargo!](#).
- `Cargo.toml` Possono essere personalizzati con profili, vedere [Personalizzazione delle build con profili di rilascio](#). Questi profili sono completamente indipendenti e indipendenti dall' AWS uso dei profili all'interno del file condiviso. AWS `config`
- Un modo comune per aggiungere dipendenze di libreria al progetto e a questo file consiste nell'utilizzare `cargo add` Per informazioni, consulta [cargo-add](#).
- Rust ha una struttura funzionale di base come la seguente. La `let` parola chiave dichiara una variabile e potrebbe essere abbinata a assignment (`=`). Se non si specifica un tipo dopo `let`, il compilatore ne dedurrà uno. Vedi [Variabili e mutabilità](#).

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

- Per dichiarare una variabile `x` con un tipo esplicito `T`, Rust utilizza la sintassi `x: T` [Vedi Tipi di dati](#).
- `struct X {}` definisce il nuovo tipo `X`. I metodi sono implementati sul tipo di struttura personalizzato `X`. I metodi per il tipo `X` sono dichiarati con blocchi di implementazione preceduti da una parola chiave `impl`. All'interno del blocco di implementazione, `self` si riferisce all'istanza della struttura su cui è stato chiamato il metodo. Vedi [Sintassi `impl` delle parole chiave e del metodo](#).
- Se è un punto esclamativo («!») segue quella che sembra essere una definizione di funzione o una chiamata di funzione, quindi il codice definisce o chiama una macro. Vedi [Macro](#).
- In Rust, gli errori irreversibili sono rappresentati dalla macro `panic!` Quando un programma incontra una, smette di funzionare, stampa un messaggio di errore, si riavvia, pulisce `panic!` lo stack e si chiude. [Vedi Errori irrecuperabili con `panic!`](#)
- Rust non supporta l'ereditarietà delle funzionalità dalle classi base come fanno gli altri linguaggi di programmazione; è così che Rust fornisce il `traits` sovraccarico dei metodi. Si potrebbe pensare che i tratti siano concettualmente simili a un'interfaccia. Tuttavia, le caratteristiche e le

interfacce reali presentano differenze e vengono spesso utilizzate in modo diverso nel processo di progettazione. Vedi [Tratti: definizione del comportamento condiviso](#).

- Il polimorfismo si riferisce al codice che supporta funzionalità per più tipi di dati senza doverli scrivere singolarmente. Rust supporta il polimorfismo tramite enumerazioni, tratti e generici. Vedi [Ereditarietà come sistema di tipi e come condivisione di codice](#).
- Rust è molto esplicito sulla memoria. I puntatori intelligenti «sono strutture di dati che agiscono come un puntatore ma hanno anche metadati e funzionalità aggiuntivi». [Vedi Smart Pointers](#).
- Il tipo Cow è un puntatore clone-on-write intelligente che aiuta a trasferire la proprietà della memoria al chiamante quando necessario. Per informazioni, consulta [Enum std::borrow::Cow](#).
- Il tipo Arc è un puntatore intelligente Atomically Reference Counted che conta le istanze allocate. Per informazioni, consulta [Struct std::sync::Arc](#).
- L'SDK per Rust utilizza spesso il pattern builder per costruire tipi complessi.

AWS SDK for Rust crea i fondamenti

- La libreria SDK for Rust è suddivisa in diverse casse di libreria ciascuna. Servizio AWS [Queste casse sono disponibili all'indirizzo https://docs.rs/](#).
- Servizio AWS le casse seguono la convenzione di denominazione di `diaws-sdk-[servicename]`, ad esempio `e. aws-sdk-s3` `aws-sdk-dynamodb`.
- La cassetta principale per la funzionalità SDK for Rust è `aws-config`. È inclusa nella maggior parte dei progetti perché fornisce funzionalità per leggere la configurazione dall'ambiente.
 - Non confondetelo con Servizio AWS quello che viene chiamato AWS Config. Poiché si tratta di un servizio, segue la convenzione standard e viene chiamato `aws-sdk-config`.

Configurazione del progetto con cui lavorare Servizi AWS

- Dovrai aggiungere una cassa al tuo progetto per ogni cassa Servizio AWS che desideri venga utilizzata dall'applicazione.
- Il modo consigliato per aggiungere una cassa è utilizzare la riga di comando nella directory del progetto eseguendo `cargo add [crateName]`, ad esempio `cargo add aws-sdk-s3`
 - Questo aggiungerà una riga nella parte `Cargo.toml` inferiore `[dependencies]` del progetto.
 - Per impostazione predefinita, questo aggiungerà la versione più recente della cassa al tuo progetto.

- Nel file sorgente, utilizzate l'istruzione per inserire gli oggetti contenuti nelle loro casse nel campo Scope. Vedi [Utilizzo di pacchetti esterni](#) sul sito Web del linguaggio di programmazione Rust.
 - I nomi delle casse sono spesso sillabati, ma i trattini vengono convertiti in caratteri di sottolineatura quando si utilizza effettivamente la cassa. Ad esempio, la `aws-config` cassa viene utilizzata nell'istruzione del codice come: `use aws_config`
- La configurazione è un argomento complesso. La configurazione può avvenire direttamente nel codice o essere specificata esternamente in variabili di ambiente o file di configurazione. Per ulteriori informazioni, consulta [Opzioni di configurazione](#).
 - Quando l'SDK carica la configurazione, vengono registrati i valori non validi anziché interrompere l'esecuzione, poiché la maggior parte delle impostazioni ha valori predefiniti ragionevoli. Per informazioni su come attivare la registrazione, consulta [Abilita la registrazione del codice AWS SDK for Rust](#)
 - La maggior parte delle variabili di ambiente e delle impostazioni dei file di configurazione vengono caricate una volta all'avvio del programma. Eventuali aggiornamenti ai valori non verranno visualizzati fino al riavvio del programma.

Runtime Tokio

- Tokio è un runtime asincrono per il linguaggio di programmazione SDK for Rust, esegue le attività. `async` Vedi [tokio.rs e docs.rs/tokio](#).
- L'SDK per Rust richiede un runtime asincrono. Ti consigliamo di aggiungere la seguente cassa ai tuoi progetti:

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

- La macro di `tokio::main` attributi crea un punto di ingresso principale asincrono al programma. Per utilizzare questa macro, aggiungetela alla riga che precede il `main` metodo, come illustrato di seguito:

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

Configura il AWS SDK for Rust

Di seguito sono riportate informazioni su come configurare AWS SDK for Rust. È necessario stabilire in che modo il codice si autentica AWS durante lo sviluppo con. Servizi AWS È inoltre necessario impostare Regione AWS quello che si desidera utilizzare.

La [AWS SDKs and Tools Reference Guide](#) contiene anche impostazioni, funzionalità e altri concetti fondamentali comuni a molti di AWS SDKs.

Argomenti

- [Creare un client di servizio](#)
- [Versioni comportamentali](#)
- [Fornitori di credenziali](#)
- [Regione AWS selezione](#)
- [Opzioni di configurazione](#)
- [Endpoint del cliente](#)
- [HTTP](#)
- [Intercettori](#)
- [Usa le funzionalità di osservabilità](#)
- [Sovrascrivi una configurazione a singola operazione del client](#)
- [Tentativi](#)
- [Timeout](#)

Creare un client di servizio

Per effettuare una richiesta a un Servizio AWS, devi prima creare un'istanza di un client per quel servizio. È possibile configurare impostazioni comuni per i client di servizio, ad esempio i timeout, il client HTTP e riprovare la configurazione.

Ogni client di servizio richiede un fornitore di credenziali Regione AWS e un provider di credenziali. L'SDK utilizza questi valori per inviare le richieste alla regione corretta per le risorse e per firmare le richieste con le credenziali corrette. Puoi specificare questi valori a livello di codice a livello di codice o caricarli automaticamente dall'ambiente.

Note

I client di servizio possono essere costosi da costruire e generalmente sono pensati per essere condivisi. Per facilitare ciò, tutte le `Client` strutture vengono implementate. `Clone`

L'SDK ha una serie di posizioni (o fonti) che controlla per trovare un valore per le impostazioni di configurazione.

1. Qualsiasi impostazione esplicita impostata nel codice o su un client di servizio stesso ha la precedenza su qualsiasi altra cosa.
2. Variabili di ambiente
 - Per i dettagli sull'impostazione delle variabili di ambiente, consultate le [variabili di ambiente nella Guida](#) di riferimento agli strumenti AWS SDKs e agli strumenti.
3. Condivisi config e credentials file
 - Per i dettagli sulla configurazione di questi file, consulta la Guida di riferimento [Condivisi configAWS SDKs e credentials file](#) in and Tools.
4. Qualsiasi valore predefinito fornito dal codice sorgente SDK stesso viene utilizzato per ultimo.
 - Alcune proprietà, come `Region`, non hanno un valore predefinito. È necessario specificarle esplicitamente nel codice, in un'impostazione di ambiente o nel `config` file condiviso. Se l'SDK non è in grado di risolvere la configurazione richiesta, le richieste API possono avere esito negativo in fase di esecuzione.

La maggior parte delle impostazioni delle variabili di ambiente `config` e delle impostazioni dei `credentials` file sono condivise da più AWS SDKs strumenti per garantire un comportamento coerente. Per visualizzare tutte le impostazioni che l'SDK può risolvere dalle variabili di ambiente o dai file di configurazione, consulta il [riferimento alle impostazioni nella Guida di riferimento](#) agli strumenti AWS SDKs e agli strumenti.

Configura un client dall'ambiente

Per creare un client con una configurazione originata dall'ambiente, usa i metodi statici della cassa: `aws-config`

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

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

La creazione di un client in questo modo è utile quando viene eseguito su Amazon Elastic Compute Cloud o in qualsiasi altro contesto in cui la configurazione di un client di servizio è disponibile direttamente dall'ambiente. AWS Lambda Questo separa il codice dall'ambiente in cui è in esecuzione e semplifica la distribuzione dell'applicazione su più utenti Regioni AWS senza modificare il codice.

È possibile sovrascrivere in modo esplicito proprietà specifiche. La configurazione esplicita ha la precedenza sulla configurazione risolta dall'ambiente di esecuzione. L'esempio seguente carica la configurazione dall'ambiente, ma sostituisce esplicitamente: Regione AWS

```
let config = aws_config::defaults(BehaviorVersion::latest())  
    .region("us-east-1")  
    .load()  
    .await;  
  
let s3 = aws_sdk_s3::Client::new(&config);
```

Note

Non tutti i valori di configurazione provengono dal client al momento della creazione. Le impostazioni relative alle credenziali, come le chiavi di accesso temporanee e la configurazione di IAM Identity Center, sono accessibili dal livello del provider di credenziali quando il client viene utilizzato per effettuare una richiesta.

Il codice `BehaviorVersion::latest()` mostrato negli esempi precedenti indica la versione dell'SDK da utilizzare per le impostazioni predefinite. `BehaviorVersion::latest()` è appropriato per la maggior parte dei casi. Per informazioni dettagliate, consultare [Versioni comportamentali](#).

Utilizza il modello builder per le impostazioni specifiche del servizio

Esistono alcune opzioni che possono essere configurate solo su un tipo di client di servizio specifico. Tuttavia, molto spesso, vorrai comunque caricare la maggior parte della configurazione dall'ambiente e quindi aggiungere specificamente le opzioni aggiuntive. Il modello builder è uno schema comune all'interno delle AWS SDK for Rust casse. Per prima cosa si carica la configurazione

generale utilizzando `aws_config::defaults`, quindi si utilizza il `from` metodo per caricare quella configurazione nel builder per il servizio con cui si sta lavorando. È quindi possibile impostare qualsiasi valore di configurazione univoco per quel servizio e quella chiamata `build`. Infine, il client viene creato da questa configurazione modificata.

```
// 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);
```

Un modo per scoprire metodi aggiuntivi disponibili per un tipo specifico di client di servizio consiste nell'utilizzare la documentazione dell'API, ad esempio per [aws_sdk_s3::config::Builder](#)

Configurazione avanzata esplicita del client

Per configurare un client di servizio con valori specifici anziché caricare una configurazione dall'ambiente, è possibile specificarli nel client Config builder come illustrato di seguito:

```
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);
```

Quando si crea una configurazione di servizio con `aws_sdk_s3::Config::builder()`, non viene caricata alcuna configurazione predefinita. I valori predefiniti vengono caricati solo quando si crea una configurazione basata su `aws_config::defaults`

Esistono alcune opzioni che possono essere configurate solo su un tipo di client di servizio specifico. L'esempio precedente mostra un esempio di ciò utilizzando la `endpoint_resolver` funzione su un client Amazon S3.

Versioni comportamentali

AWS SDK for Rust gli sviluppatori si aspettano e fanno affidamento sul comportamento robusto e prevedibile offerto dal linguaggio e dalle sue principali librerie. Per aiutare gli sviluppatori che utilizzano l'SDK per Rust a ottenere il comportamento previsto, le configurazioni dei client devono includere un `BehaviorVersion` `BehaviorVersion` specifica la versione dell'SDK di cui sono previste le impostazioni predefinite. Ciò consente all'SDK di evolversi nel tempo, modificando le migliori pratiche per soddisfare nuovi standard e supportare nuove funzionalità senza impatti negativi imprevisti sul comportamento dell'applicazione.

Warning

Se provate a configurare l'SDK o a creare un client senza specificare esplicitamente `a`, il costruttore lo farà `BehaviorVersion panic`.

Ad esempio, immaginate che venga rilasciata una nuova versione dell'SDK con una nuova politica di riprova predefinita. Se l'applicazione utilizza una versione `BehaviorVersion` corrispondente a una versione precedente dell'SDK, viene utilizzata tale configurazione precedente anziché la nuova configurazione predefinita.

Ogni volta che viene rilasciata una nuova versione comportamentale dell'SDK per Rust, la precedente `BehaviorVersion` viene contrassegnata con l'`deprecated` attributo SDK for Rust e viene aggiunta la nuova versione. Ciò fa sì che vengano visualizzati degli avvisi in fase di compilazione, ma per il resto consente alla compilazione di continuare come al solito. `BehaviorVersion::latest()` viene inoltre aggiornato per indicare il comportamento predefinito della nuova versione.

Note

Nella maggior parte dei casi, è necessario utilizzare `BehaviorVersion::latest()` nel codice o il flag di funzionalità `behavior-version-latest` nel `Cargo.toml` file. Si consiglia di aggiungere una versione specifica solo per il tempo necessario.

Imposta la versione del comportamento in `Cargo.toml`

Puoi specificare la versione di comportamento per l'SDK e i singoli moduli, ad esempio `aws-sdk-s3` o `aws-sdk-iam`, includendo un flag di funzionalità appropriato nel `Cargo.toml` file. Al momento, solo la `latest` versione dell'SDK è supportata in: `Cargo.toml`

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

Imposta la versione del comportamento nel codice

Il codice può modificare la versione del comportamento in base alle esigenze specificandola durante la configurazione dell'SDK o di un client:

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

Questo esempio crea una configurazione che utilizza l'ambiente per configurare l'SDK ma imposta su `BehaviorVersion v2023_11_09()`

Fornitori di credenziali

Per effettuare richieste di AWS utilizzo di AWS SDK for Rust, l'SDK utilizza credenziali con firma crittografica emesse da AWS. In fase di esecuzione, l'SDK recupera i valori di configurazione delle credenziali controllando diverse posizioni.

Se la configurazione recuperata include [impostazioni di accesso AWS IAM Identity Center Single Sign-On](#), l'SDK collabora con IAM Identity Center per recuperare le credenziali temporanee utilizzate per effettuare la richiesta. Servizi AWS

Se la configurazione recuperata include [credenziali temporanee](#), l'SDK le utilizza per effettuare chiamate. Servizio AWS Le credenziali temporanee sono costituite da chiavi di accesso e un token di sessione.

L'autenticazione con AWS può essere gestita all'esterno del codebase. Molti metodi di autenticazione possono essere rilevati, utilizzati e aggiornati automaticamente dall'SDK utilizzando la catena di fornitori di credenziali.

Per le opzioni guidate su come iniziare a utilizzare AWS l'autenticazione del progetto, consultate [Autenticazione e accesso](#) nella Guida di riferimento agli strumenti AWS SDKs e agli strumenti.

La catena di fornitori di credenziali

Se non specificate esplicitamente un fornitore di credenziali durante la creazione di un client, l'SDK per Rust utilizza una catena di fornitori di credenziali che controlla una serie di punti in cui è possibile fornire le credenziali. Una volta che l'SDK trova le credenziali in una di queste posizioni, la ricerca si interrompe. Per i dettagli sulla creazione di client, consulta [Creare un client di servizio](#)

L'esempio seguente non specifica un fornitore di credenziali nel codice. L'SDK utilizza la catena di provider di credenziali per rilevare l'autenticazione che è stata configurata nell'ambiente di hosting e utilizza tale autenticazione per le chiamate verso. Servizi AWS

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

Ordine di recupero delle credenziali

La catena di fornitori di credenziali cerca le credenziali utilizzando la seguente sequenza predefinita:

1. Accedi alle variabili di ambiente chiave

L'SDK tenta di caricare le credenziali dalle variabili `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` di `AWS_SESSION_TOKEN` ambiente.

2. I file condivisi AWS **config** e **credentials**

L'SDK tenta di caricare le credenziali dal `[default]` profilo nei file condivisi AWS `config` e `credentials`. Puoi utilizzare la variabile di `AWS_PROFILE` ambiente per scegliere un profilo denominato che desideri venga caricato dall'SDK anziché utilizzare. `[default]` I `credentials` file `config` and sono condivisi da vari AWS SDKs strumenti. Per ulteriori informazioni su questi file, consulta la Guida di riferimento [Condivisi configAWS SDKs e credentials file](#) in the and Tools.

Se utilizzi IAM Identity Center per l'autenticazione, questo è quando l'SDK per Rust utilizza il token Single Sign-on che è stato impostato eseguendo il comando CLI. `AWS aws sso login` L'SDK utilizza le credenziali temporanee scambiate da IAM Identity Center con un token valido. L'SDK utilizza quindi le credenziali temporanee quando chiama. Servizi AWS Per informazioni dettagliate su questo processo, consulta [Understand SDK per la risoluzione delle credenziali Servizi AWS](#) nella AWS SDKs and Tools Reference Guide.

- Per indicazioni sulla configurazione di questo provider, consulta l'[autenticazione di IAM Identity Center](#) nella AWS SDKs and Tools Reference Guide.

- Per i dettagli sulle proprietà di configurazione dell'SDK per questo provider, consulta il provider di [credenziali IAM Identity Center](#) nella AWS SDKs and Tools Reference Guide.

3. AWS STS identità web

Quando si creano applicazioni mobili o applicazioni Web basate su client che richiedono l'accesso a AWS, AWS Security Token Service (AWS STS) restituisce un set di credenziali di sicurezza temporanee per gli utenti federati autenticati tramite un provider di identità pubblica (IdP).

- Quando lo specificate in un profilo, l'SDK o lo strumento tenta di recuperare le credenziali temporanee utilizzando il metodo API `AssumeRoleWithWebIdentity`. Per i dettagli su questo metodo, consulta l'API [AssumeRoleWithWebIdentity](#) Reference. AWS Security Token Service
- Per indicazioni sulla configurazione di questo provider, consulta [Federate with web identity o OpenID Connect](#) nella AWS SDKs and Tools Reference Guide.
- Per i dettagli sulle proprietà di configurazione SDK per questo provider, consulta [Assume il ruolo del provider di credenziali nella and Tools Reference](#) Guide. AWS SDKs

4. Credenziali dei container Amazon ECS e Amazon EKS

Alle attività di Amazon Elastic Container Service e agli account di servizio Kubernetes può essere associato un ruolo IAM. Le autorizzazioni concesse nel ruolo IAM vengono assunte dai contenitori in esecuzione nell'attività o nei contenitori del pod. Questo ruolo consente al codice dell'applicazione SDK for Rust (sul contenitore) di utilizzarne altri. Servizi AWS

L'SDK tenta di recuperare le credenziali dalle variabili di ambiente `AWS_CONTAINER_CREDENTIALS_FULL_URI` o `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI`, che possono essere impostate automaticamente da Amazon ECS e Amazon EKS.

- Per dettagli sulla configurazione di questo ruolo per Amazon ECS, consulta il [ruolo IAM delle attività di Amazon ECS](#) nella Amazon Elastic Container Service Developer Guide.
- Per informazioni sulla configurazione di Amazon EKS, consulta [Configurazione dell'agente Amazon EKS Pod Identity](#) nella Guida per l'utente di Amazon EKS.
- Per i dettagli sulle proprietà di configurazione SDK per questo provider, consulta [Container credential provider](#) nella AWS SDKs and Tools Reference Guide.

5. Servizio di metadati Amazon EC2 Instance

Crea un ruolo IAM e collegalo alla tua istanza. L'applicazione SDK for Rust sull'istanza tenta di recuperare le credenziali fornite dal ruolo dai metadati dell'istanza.

- L'SDK per Rust supporta solo. [IMDSv2](#)
 - Per i dettagli sulla configurazione di questo ruolo e sull'utilizzo dei metadati, consulta i [metadati IAM roles for Amazon EC2 e Work with Instance](#) nella Amazon EC2 User Guide.
 - Per i dettagli sulle proprietà di configurazione SDK per questo provider, consulta il provider di [credenziali IMDS nella and Tools Reference](#) Guide.AWS SDKs
6. Se le credenziali non sono ancora state risolte a questo punto, l'operazione panics con un errore.

Per i dettagli sulle impostazioni di configurazione dei provider di AWS credenziali, consulta [Fornitori di credenziali standardizzati](#) nel riferimento alle impostazioni della AWS SDKs and Tools Reference Guide.

Fornitore di credenziali esplicito

Invece di affidarti alla catena di fornitori di credenziali per rilevare il metodo di autenticazione, puoi specificare un provider di credenziali specifico che l'SDK deve utilizzare. Quando carichi la configurazione generale utilizzando `aws_config::defaults`, puoi specificare un provider di credenziali personalizzato come mostrato di seguito:

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

È possibile implementare il proprio provider di credenziali implementando la [ProvideCredentials](#) caratteristica.

Memorizzazione nella cache delle identità

L'SDK memorizzerà nella cache le credenziali e altri tipi di identità come i token SSO. Per impostazione predefinita, l'SDK utilizza un'implementazione lazy cache che carica le credenziali alla prima richiesta, le memorizza nella cache e quindi tenta di aggiornarle durante un'altra richiesta quando stanno per scadere. I client creati dalla stessa condivideranno un. `SdkConfig IdentityCache`

Regione AWS selezione

È possibile accedere a Servizi AWS ciò che opera in un'area geografica specifica utilizzando Regioni AWS. Ciò può essere utile sia per la ridondanza sia per mantenere attivi i dati e le applicazioni vicino a dove voi e i vostri utenti vi accedete. Per ulteriori informazioni su come vengono utilizzate le regioni, consulta [Regione AWS](#) la Guida di riferimento agli strumenti AWS SDKs e agli strumenti.

Important

La maggior parte delle risorse risiede in una regione specifica Regione AWS ed è necessario fornire la regione corretta per la risorsa quando si utilizza l'SDK.

Regione AWS catena di fornitori

Il seguente processo di ricerca viene utilizzato quando si carica la configurazione di un client di servizio dall'ambiente di esecuzione. Il primo valore che l'SDK trova impostato viene utilizzato nella configurazione del client. Per ulteriori informazioni sulla creazione di client di servizio, consulta [Configura un client dall'ambiente](#).

1. Qualsiasi regione esplicita impostata a livello di codice.
2. La variabile di ambiente `AWS_REGION` è selezionata.
 - Se si utilizza il AWS Lambda servizio, questa variabile di ambiente viene impostata automaticamente dal contenitore. AWS Lambda
3. La `region` proprietà nel `AWS config` file condiviso viene verificata.
 - La variabile di `AWS_CONFIG_FILE` ambiente può essere utilizzata per modificare la posizione del `config` file condiviso. Per ulteriori informazioni sulla posizione in cui è conservato questo file, consulta [Posizione del file condiviso config e dei credentials file](#) nella Guida di riferimento agli strumenti AWS SDKs e agli strumenti.
 - La variabile di `AWS_PROFILE` ambiente può essere utilizzata per selezionare un profilo denominato anziché quello predefinito. Per ulteriori informazioni sulla configurazione di diversi profili, consulta [configShared and credentials files nella AWS SDKs and Tools Reference Guide](#).
4. L'SDK tenta di utilizzare Amazon EC2 Instance Metadata Service per determinare la regione dell'istanza Amazon EC2 attualmente in esecuzione.
 - Gli AWS SDK for Rust unici supporti. IMDSv2

Opzioni di configurazione

Molte impostazioni di configurazione possono essere gestite al di fuori del codice. La maggior parte delle impostazioni di configurazione può essere impostata come variabili di ambiente o in un AWS config file condiviso separato. Il config file condiviso può mantenere set di impostazioni separati, chiamati profili, per fornire configurazioni diverse per ambienti o test diversi.

Le variabili di ambiente e le impostazioni dei config file condivisi sono standardizzate e condivise tra AWS SDKs strumenti per supportare funzionalità coerenti in diverse lingue.

Consulta la Guida di riferimento agli strumenti AWS SDKs e scopri come configurare l'applicazione con questi metodi, oltre ai dettagli su ciascuna impostazione cross-sdk. Alcune pagine di interesse potrebbero essere:

- [Condivisi config e credentials file](#): spiega i profili e il formato di questi file di configurazione.
- [Posizione dei credentials file condivisi config e](#): spiega la posizione predefinita di questi file e come modificarla.
- [Supporto per le variabili di ambiente](#): spiega l'impostazione delle variabili di ambiente.
- [Riferimento alle impostazioni](#): informazioni di riferimento su tutte le impostazioni Cross-SDK.

Endpoint del cliente

Quando AWS SDK for Rust chiama un Servizio AWS, uno dei primi passi è determinare dove indirizzare la richiesta. Questo processo è noto come risoluzione degli endpoint.

Puoi configurare la risoluzione degli endpoint per l'SDK quando crei un client di servizio. La configurazione predefinita per la risoluzione degli endpoint in genere va bene, ma ci sono diversi motivi per cui potresti voler modificare la configurazione predefinita. Di seguito sono riportati due esempi di motivi:

- Per effettuare richieste a una versione non definitiva di un servizio o a una distribuzione locale di un servizio.
- Per accedere a funzionalità di servizio specifiche non ancora modellate nell'SDK.

⚠ Warning

La risoluzione degli endpoint è un argomento SDK avanzato. Se modifichi le impostazioni predefinite, rischi di violare il codice. Le impostazioni predefinite si applicano alla maggior parte degli utenti negli ambienti di produzione.

Gli endpoint personalizzati possono essere impostati globalmente in modo da essere utilizzati per tutte le richieste di servizio oppure è possibile impostare un endpoint personalizzato per uno specifico. Servizio AWS

Gli endpoint personalizzati possono essere configurati utilizzando variabili o impostazioni di ambiente nel file condiviso. AWS config Per informazioni su questo approccio, consulta [Endpoint specifici del servizio](#) nella Guida di riferimento agli strumenti AWS SDKs e agli strumenti. Per l'elenco completo delle impostazioni dei config file condivisi e delle variabili di ambiente per tutti Servizi AWS, consulta [Identificatori](#) per endpoint specifici del servizio.

In alternativa, questa personalizzazione può essere configurata anche nel codice, come illustrato nelle sezioni seguenti.

Configurazione personalizzata

È possibile personalizzare la risoluzione degli endpoint di un client di servizio con due metodi disponibili al momento della creazione del client:

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

È possibile impostare entrambe le proprietà. Tuttavia, molto spesso ne fornisci solo una. Per uso generale, `endpoint_url` è più frequentemente personalizzato.

Imposta l'URL dell'endpoint

È possibile impostare un valore `endpoint_url` per indicare un nome host «base» per il servizio. Questo valore, tuttavia, non è definitivo poiché viene passato come parametro all'`ResolveEndpoint`istanza del client. L'`ResolveEndpoint`implementazione può quindi ispezionare e potenzialmente modificare quel valore per determinare l'endpoint finale.

Imposta il resolver degli endpoint

`ResolveEndpoint` L'implementazione di un client di servizio determina l'endpoint risolto finale che l'SDK utilizza per una determinata richiesta. Un client di servizio chiama il `resolve_endpoint` metodo per ogni richiesta e utilizza il [EndpointFuture](#) valore restituito dal resolver senza ulteriori modifiche.

L'esempio seguente dimostra la fornitura di un'implementazione di endpoint resolver personalizzata per un client Amazon S3 che risolve un endpoint diverso per fase, come lo staging e la produzione:

```
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

I resolver per endpoint, e per estensione la `ResolveEndpoint` caratteristica, sono specifici per ogni servizio e quindi possono essere configurati solo nella configurazione del client del servizio. L'URL dell'endpoint, d'altra parte, può essere configurato utilizzando

la configurazione condivisa (applicabile a tutti i servizi da essa derivati) o per un servizio specifico.

ResolveEndpoint parametri

Il `resolve_endpoint` metodo accetta parametri specifici del servizio che contengono le proprietà utilizzate nella risoluzione degli endpoint.

Ogni servizio include le seguenti proprietà di base:

Nome	Tipo	Descrizione
<code>region</code>	Stringa	Del cliente Regione AWS
<code>endpoint</code>	Stringa	Una rappresentazione in formato stringa del set di valori di <code>endpointUrl</code>
<code>use_fips</code>	Booleano	Se gli endpoint FIPS sono abilitati nella configurazione del client
<code>use_dual_stack</code>	Booleano	Se gli endpoint dual-stack sono abilitati nella configurazione del client

Servizi AWS può specificare proprietà aggiuntive richieste per la risoluzione. Ad esempio, i [parametri degli endpoint](#) di Amazon S3 includono il nome del bucket e anche diverse impostazioni di funzionalità specifiche di Amazon S3. Ad esempio, la `force_path_style` proprietà determina se è possibile utilizzare l'indirizzamento dell'host virtuale.

Se si implementa il proprio provider, non dovrebbe essere necessario creare un'istanza personalizzata dei parametri degli endpoint. L'SDK fornisce le proprietà per ogni richiesta e le trasmette all'implementazione di `resolve_endpoint`

Confronta l'utilizzo con `endpoint_url` l'utilizzo `endpoint_resolver`

È importante comprendere che le due configurazioni seguenti, una che utilizza `endpoint_url` e l'altra che utilizza `endpoint_resolver`, NON producono client con un comportamento di risoluzione degli endpoint equivalente.

```

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();

```

Il client che imposta il `endpoint_url` specifica un URL di base che viene passato al provider (predefinito), che può essere modificato come parte della risoluzione degli endpoint.

Il client che imposta il `endpoint_resolver` specifica l'URL finale utilizzato dal client Amazon S3.

Esempi

Gli endpoint personalizzati vengono spesso utilizzati per i test. Invece di effettuare chiamate verso il servizio basato sul cloud, le chiamate vengono indirizzate a un servizio simulato ospitato localmente.

Due di queste opzioni sono:

- [DynamoDB](#) local: una versione locale del servizio Amazon DynamoDB.
- [LocalStack](#)— un emulatore di servizi cloud che viene eseguito in un contenitore sul computer locale.

Gli esempi seguenti illustrano due modi diversi per specificare un endpoint personalizzato per utilizzare queste due opzioni di test.

Usa DynamoDB local direttamente nel codice

Come descritto nelle sezioni precedenti, è possibile impostare `endpoint_url` direttamente nel codice l'override dell'endpoint di base in modo che punti al server DynamoDB locale. Nel tuo codice:

```
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);
```

L'[esempio completo](#) è disponibile su GitHub.

Usa LocalStack usando il **config** file

Puoi impostare [endpoint specifici del servizio](#) nel tuo file condiviso. AWS config I seguenti set di profili di configurazione `endpoint_url` a cui connettersi localhost sulla porta. 4566 Per ulteriori informazioni sulla LocalStack configurazione, consulta [Accesso LocalStack tramite l'URL dell'endpoint sul sito Web](#) dei LocalStack documenti.

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

L'SDK raccoglierà le modifiche nel config file condiviso e le applicherà ai client SDK quando utilizzi il profilo. `localstack` Utilizzando questo approccio, il codice non deve includere alcun riferimento agli endpoint e avrebbe il seguente aspetto:

```
// 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);
```

L'[esempio completo](#) è disponibile su. GitHub

HTTP

L' AWS SDK per Rust fornisce le seguenti configurazioni relative a HTTP:

Argomenti

- [Hyper 1.x](#)

Hyper 1.x

Note

AWS SDK for Rust È stato sviluppato prima del `hyper` rilascio di una versione 1.0 stabile. L'SDK attualmente utilizza per impostazione predefinita la versione v0.14 di `hyper`

`hyper` è una libreria HTTP di livello inferiore per Rust che può essere utilizzata con per effettuare chiamate ai servizi API. AWS SDK for Rust Per la documentazione di riferimento su questa cassa, vedi [hyper](#) - Rust. Per impostazione predefinita, l'SDK per Rust utilizza la versione v0.14 di e un provider. `hyper` `CryptoMode:Ring`

I client con supporto `hyper` 1.0 si trovano nella cassa. [aws-smithy-experimental](#)

Scegli un provider di criptovalute

Devi scegliere `CryptoProvider` e abilitare la funzionalità richiesta nella `aws-smithy-experimental` cassa.

Sono disponibili le seguenti `CryptoProvider` modalità:

- `CryptoMode:AwsLc` - un provider crittografico basato su. [aws-lc-rs](#) Si tratta di una libreria crittografica generica gestita dal team di AWS crittografia AWS e dai suoi clienti, basata sul codice del progetto Google BoringSSL e del progetto OpenSSL.

- Richiede funzionalità. `crypto-aws-lc`
- `CryptoMode:AwsLcFips`- un provider di crittografia conforme a FIPS basato su. [aws-lc-rs](#)
- `crypto-aws-lc-fips` Richiede funzionalità.

Note

Ciò potrebbe richiedere strumenti di compilazione aggiuntivi per poter compilare. Per ulteriori informazioni, consulta il [aws-lc-rs](#) repository e le istruzioni di compilazione.

- `CryptoMode:Ring`- un provider di crittografia basato su. [ring](#) Questa è una «criptovaluta sicura, veloce e di piccole dimensioni che utilizza Rust».
- Richiede funzionalità `crypto-ring`.

Note

Questo è il provider di crittografia utilizzato dall'SDK per Rust nel client `hyper` v0.14 predefinito.

Come abilitare una funzionalità per una cassa

L'esempio seguente mostra come aggiungere la `crypto-aws-lc` funzionalità alla `aws-smithy-experimental` cassa del progetto: `Cargo.toml`

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

In alternativa, è possibile eseguire il comando seguente al prompt dei comandi nella cartella del progetto:

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

Per ulteriori informazioni, consultate [Funzionalità](#) del The Cargo Book.

Crea un client **hyper** 1.0 e usalo con l'SDK

Per creare un client `hyper` 1.0 `Cargo.toml`, aggiungi la `aws-smithy-experimental` cassa nel tuo progetto. `HyperClientBuilder` richiede che tu scelga un `CryptoProvider`. La sezione

precedente descriveva le scelte del provider e come abilitare una delle funzionalità relative alle criptovalute del `aws-smithy-experimental` crate.

Il codice seguente utilizza `CryptoMode::AwsLc` come provider di crittografia un client HTTP hyper 1.0 da utilizzare con un client di servizio Amazon Simple Storage Service (Amazon S3). Per questo esempio, il tuo progetto deve includere `Cargo.toml` anche Servizio AWS la cassa, `aws-sdk-s3`

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

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

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

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

Il client HTTP può essere impostato sulla struttura di configurazione condivisa o sulla struttura di configurazione di un servizio specifico.

Intercettori

È possibile utilizzare gli intercettori per agganciarsi all'esecuzione di richieste e risposte API. Gli intercettori sono meccanismi aperti in cui l'SDK richiama il codice scritto dall'utente per inserire il comportamento nel ciclo di vita della richiesta/risposta. In questo modo, puoi modificare una richiesta in corso, eseguire il debug dell'elaborazione delle richieste, visualizzare gli errori e altro ancora.

L'esempio seguente mostra un semplice intercettore che aggiunge un'intestazione aggiuntiva a tutte le richieste in uscita prima che venga inserito il ciclo di ripetizione:

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

[Per ulteriori informazioni e per conoscere gli hook di intercettazione disponibili, vedete il tratto `Intercept`.](#)

Registrazione Interceptor

Gli intercettori vengono registrati quando si crea un client di servizio o quando si sostituisce la configurazione per un'operazione specifica. La registrazione è diversa a seconda che si desideri che l'interceptor si applichi a tutte le operazioni per il cliente o solo a quelle specifiche.

Interceptor per tutte le operazioni su un client di servizio

Per registrare un intercettore per l'intero client, aggiungete l'intercettore utilizzando il pattern.

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 solo per un'operazione specifica

Per registrare un intercettore per una sola operazione, utilizzate l'estensione. `customize` È possibile sovrascrivere le configurazioni dei client di servizio a livello di singola operazione utilizzando questo metodo. Per ulteriori informazioni sulle operazioni personalizzabili, vedere. [Sovrascrivi una configurazione a singola operazione del client](#)

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

Usa le funzionalità di osservabilità

L'osservabilità è la misura in cui lo stato attuale di un sistema può essere dedotto dai dati che emette. I dati emessi vengono comunemente definiti telemetria.

Argomenti

- [Abilita la registrazione del codice AWS SDK for Rust](#)

Abilita la registrazione del codice AWS SDK for Rust

AWS SDK for Rust Utilizza il framework di [tracciamento](#) per la registrazione.

1. In un prompt dei comandi per la directory del progetto, aggiungi [tracing-subscriber](#) crate come dipendenza:

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

Questo aggiunge la cassa alla sezione del file. [dependencies] Cargo.toml

2. Inizializza il sottoscrittore. Di solito questa operazione viene eseguita all'inizio della main funzione prima di chiamare qualsiasi operazione SDK for Rust:

```
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. Attiva la registrazione utilizzando la variabile di RUST_LOG ambiente. Per abilitare la visualizzazione delle informazioni di registrazione, in un prompt dei comandi, impostate la variabile di RUST_LOG ambiente sul livello a cui desiderate effettuare il login. L'esempio seguente imposta la registrazione sul livello: debug

Linux/macOS

```
$ RUST_LOG=debug
```

Windows

Se si utilizza VSCode, la finestra del terminale spesso è predefinita su PowerShell. Verifica il tipo di prompt che stai utilizzando.

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

PowerShell

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

4. Esegui il programma:

```
$ cargo run
```

Dovresti vedere un output aggiuntivo nella finestra della console o del terminale.

Per ulteriori informazioni, consulta [Filtrare gli eventi con variabili di ambiente](#) nella tracing-subscriber documentazione.

Interpreta l'output del registro

Dopo aver attivato la registrazione seguendo i passaggi della sezione precedente, le informazioni di registro aggiuntive verranno stampate in modo predefinito.

Se si utilizza il formato di output del registro predefinito (chiamato «completo» dal modulo di tracciamento), le informazioni visualizzate nell'output del registro sono simili alle seguenti:

```
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
```

Ogni voce include quanto segue:

- Il timestamp della voce di registro.
- Il livello di registro della voce. Si tratta di una parola come INFODEBUG, oTRACE.
- L'insieme annidato di [intervalli da](#) cui è stata generata la voce di registro, separati da due punti («:»). Ciò consente di identificare l'origine della voce di registro.
- Il percorso del modulo Rust contenente il codice che ha generato la voce di registro.
- Il testo del messaggio di log.

I formati di output standard del modulo di tracciamento utilizzano codici di escape ANSI per colorare l'output. Tenete a mente queste sequenze di escape quando filtrate o cercate l'output.

Note

All'interno del `sdk_invocation_id` set annidato di intervalli viene visualizzato un ID univoco generato dal client SDK per facilitare la correlazione dei messaggi di registro. Non è correlato all'ID della richiesta trovato nella risposta di un. Servizio AWS

Ottimizza i tuoi livelli di registrazione

Se utilizzi una cassa che supporta il filtraggio di un ambiente, ad esempio `tracing_subscriber`, puoi controllare la verbosità dei log per modulo.

Puoi attivare lo stesso livello di registrazione per ogni modulo. Quanto segue imposta il livello di registrazione su ogni `trace` modulo:

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

È possibile attivare la registrazione a livello di traccia per un modulo specifico. Nell'esempio seguente, solo i log provenienti da `aws_smithy_runtime` verranno inseriti al livello. `trace`

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

È possibile specificare un livello di registro diverso per più moduli separandoli con virgole. L'esempio seguente imposta il `aws_config` modulo sulla registrazione dei `trace` livelli e il `aws_smithy_runtime` modulo sulla registrazione dei livelli. `debug`

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

La tabella seguente descrive alcuni dei moduli che è possibile utilizzare per filtrare i messaggi di registro:

Prefix	Descrizione
<code>aws_smithy_runtime</code>	Registrazione via cavo di richiesta e risposta
<code>aws_config</code>	Caricamento delle credenziali
<code>aws_sigv4</code>	Richieste di firma e richieste canoniche

Un modo per capire quali moduli è necessario includere nell'output di registro è registrare prima tutto, quindi trovare il nome della cassa nell'output del registro per le informazioni necessarie. Quindi puoi impostare la variabile di ambiente di conseguenza ed eseguire nuovamente il programma.

Sovrascrivi una configurazione a singola operazione del client

Dopo aver [creato un client di servizio](#), la configurazione diventa immutabile e verrà applicata a tutte le operazioni successive. Sebbene la configurazione non possa essere modificata a questo punto, può essere sostituita in base all'operazione.

Ogni Operation Builder dispone di un `customize` metodo per creare un file `CustomizableOperation` in modo da poter sovrascrivere una singola copia della configurazione esistente. La configurazione originale del client rimarrà invariata.

L'esempio seguente mostra la creazione di un client Amazon S3 che chiama due operazioni, la seconda delle quali viene sovrascritta per l'invio a un'altra. Regione AWS Tutte le chiamate agli oggetti di Amazon S3 utilizzano la us-east-1 regione tranne quando la chiamata API viene esplicitamente sovrascritta per utilizzare la modifica. 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

L'esempio precedente riguarda Amazon S3, tuttavia il concetto è lo stesso per tutte le operazioni. Alcune operazioni potrebbero avere metodi aggiuntivi attivi. `CustomizableOperation`

Per un esempio di aggiunta di un intercettore utilizzato `customize` per una singola operazione, vedere. [Interceptor solo per un'operazione specifica](#)

Tentativi

AWS SDK for Rust Fornisce un comportamento di ripetizione predefinito e opzioni di configurazione personalizzabili. Chiamate per restituire Servizi AWS occasionalmente eccezioni impreviste. Alcuni tipi di errori, come gli errori di limitazione o gli errori transitori, potrebbero avere esito positivo se la chiamata viene ritentata.

Il comportamento dei tentativi può essere configurato globalmente utilizzando le variabili o le impostazioni di ambiente nel file condiviso. `AWS config` Per informazioni su questo approccio, consulta [Retry behavior nella AWS SDKs and Tools Reference](#) Guide. Include anche informazioni dettagliate sulle implementazioni della strategia Retry e su come sceglierne una piuttosto che un'altra.

In alternativa, queste opzioni possono essere configurate anche nel codice, come illustrato nelle sezioni seguenti.

Configurazione predefinita dei nuovi tentativi

Ogni client di servizio utilizza per impostazione predefinita la configurazione della strategia di `standard riprova` fornita tramite la struttura. [RetryConfig](#) Per impostazione predefinita, una chiamata verrà tentata tre volte (il tentativo iniziale, più due tentativi). Inoltre, ogni nuovo tentativo verrà posticipato di una breve durata casuale per evitare tempeste di tentativi. Questa convenzione è adatta per la maggior parte dei casi d'uso, ma potrebbe non essere adatta in circostanze specifiche, come i sistemi ad alta produttività.

Solo alcuni tipi di errori sono considerati correggibili da. SDKs Esempi di errori riutilizzabili sono:

- timeout del socket
- limitazione sul lato dei servizi
- errori di servizio transitori come le risposte HTTP 5XX

I seguenti esempi non sono considerati riutilizzabili:

- Parametri mancanti o non validi
- errori di autenticazione/sicurezza
- eccezioni di configurazione errata

È possibile personalizzare la strategia di standard riprova impostando il numero massimo di tentativi, ritardi e configurazione di backoff.

Numero massimo di tentativi

Puoi personalizzare il numero massimo di tentativi nel tuo codice fornendo una modifica

[RetryConfig](#)al tuo `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;
```

Ritardi e arretramenti

Se è necessario un nuovo tentativo, la strategia di riprova predefinita attende prima di effettuare il tentativo successivo. Il ritardo per il primo tentativo è piccolo, ma aumenta esponenzialmente per i tentativi successivi. La quantità massima di ritardo è limitata in modo che non diventi troppo grande.

Il jitter casuale viene applicato ai ritardi tra tutti i tentativi. Il jitter aiuta a mitigare l'effetto delle flotte di grandi dimensioni che possono causare tempeste di ritardi. [Per una discussione più approfondita su backoff e jitter esponenziali, vedete Exponential Backoff and Jitter nel blog di architettura.AWS](#)

Puoi personalizzare le impostazioni di ritardo nel tuo codice fornendo una modifica al tuo.

[RetryConfig](#)al tuo `aws_config::defaults` Il codice seguente imposta la configurazione in modo da ritardare il primo tentativo fino a 100 millisecondi e che il tempo massimo tra un tentativo e l'altro sia di 5 secondi.

```
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;
```

Modalità di riprova adattiva

In alternativa alla strategia Mode Retry, la standard adaptive Mode Retry Strategy è un approccio avanzato che punta alla frequenza di richiesta ideale per ridurre al minimo gli errori di throttling.

Note

Adaptive retry è una modalità di riprova avanzata. L'utilizzo di questa strategia in genere non è consigliato. Vedi [Comportamento dei tentativi](#) nella Guida di riferimento agli strumenti AWS SDKs e agli strumenti.

I tentativi adattivi includono tutte le funzionalità dei nuovi tentativi standard. Aggiunge un limitatore di velocità sul lato client che misura la frequenza delle richieste limitate rispetto alle richieste non limitate. Inoltre limita il traffico per cercare di rimanere entro una larghezza di banda sicura, evitando idealmente errori di limitazione.

La tariffa si adatta in tempo reale alle mutevoli condizioni di servizio e ai modelli di traffico e potrebbe aumentare o diminuire di conseguenza la velocità di traffico. In modo critico, il limitatore di velocità potrebbe ritardare i tentativi iniziali in scenari di traffico intenso.

È possibile selezionare la strategia di adaptive riprova nel codice fornendo una modifica:

[RetryConfig](#)

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

Timeout

AWS SDK for Rust Fornisce diverse impostazioni per la gestione dei timeout delle richieste e dei flussi di dati in stallo. Queste aiutano l'applicazione a comportarsi in modo ottimale quando si verificano ritardi e guasti imprevisti nella rete.

Timeout delle API

Quando si verificano problemi temporanei che possono causare tempi lunghi o fallire completamente, è importante rivedere e impostare i timeout in modo che l'applicazione possa fallire rapidamente e si comporti in modo ottimale. Le richieste che hanno esito negativo possono essere ritentate automaticamente dall'SDK. È buona norma impostare dei timeout sia per il singolo tentativo che per l'intera richiesta.

L'SDK per Rust fornisce un timeout predefinito per stabilire una connessione per una richiesta. L'SDK non ha alcun tempo di attesa massimo predefinito impostato per la ricezione di una risposta per un tentativo di richiesta o per l'intera richiesta. Sono disponibili le seguenti opzioni di timeout:

Parametro	Valore predefinito	Descrizione
Timeout Connect	3,1 secondi	Il tempo massimo di attesa per stabilire una connessione prima di rinunciare.
Timeout dell'operazione	Nessuno	Il tempo massimo di attesa prima di ricevere una risposta dall'SDK per Rust, inclusi tutti i nuovi tentativi.
Timeout del tentativo di operazione	Nessuno	Il tempo massimo di attesa per un singolo tentativo HTTP, dopo il quale è possibile ritentare la chiamata API.
Timeout di lettura	Nessuno	Il tempo massimo di attesa per leggere il primo byte di una risposta dal momento in cui viene avviata la richiesta.

L'esempio seguente mostra la configurazione di un client Amazon S3 con valori di timeout personalizzati:

```
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);
```

Quando utilizzi contemporaneamente il timeout dell'operazione e quello dei tentativi, imposti un limite rigido al tempo totale impiegato per tutti i tentativi tra più tentativi. È inoltre possibile impostare una richiesta HTTP individuale in modo che abbia esito negativo rapidamente in caso di richiesta lenta.

In alternativa all'impostazione di questi valori di timeout sul client del servizio per tutte le operazioni, puoi configurarli o [sostituirli per una singola](#) richiesta.

Important

I timeout delle operazioni e dei tentativi non si applicano ai dati di streaming consumati dopo che l'SDK per Rust ha restituito una risposta. Ad esempio, il consumo di dati da un `ByteStream` membro di una risposta non è soggetto a timeout operativi.

Protezione del flusso bloccata

L'SDK per Rust fornisce un'altra forma di timeout relativa al rilevamento di flussi in stallo. Uno stream in stallo è un flusso di upload o download che non produce dati per un periodo di prova superiore a quello configurato. Questo aiuta a evitare che le applicazioni si blocchino a tempo indeterminato e non facciano mai progressi.

La protezione dello streaming in stallo restituirà un errore quando uno stream rimane inattivo per un periodo superiore al periodo accettabile.

Per impostazione predefinita, l'SDK per Rust abilita la protezione dallo streaming in fase di stallo sia per i caricamenti che per i download e rileva almeno 1 byte/sec di attività con un generoso periodo di prova di 20 secondi.

L'esempio seguente mostra una soluzione personalizzata `StalledStreamProtectionConfig` che disabilita la protezione dal caricamento e modifica il periodo di prova per l'assenza di attività a 10 secondi:

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

```
    StalledStreamProtectionConfig::enabled()
        .upload_enabled(false)
        .grace_period(Duration::from_secs(10))
        .build()
)
.load()
.await;
```

Warning

Stalled Stream Protection è un'opzione di configurazione avanzata. Consigliamo di modificare questi valori solo se l'applicazione richiede prestazioni più elevate o se ciò causa altri problemi.

Disattiva la protezione dello streaming in stallo

L'esempio seguente mostra come disabilitare completamente la protezione dai flussi in stallo:

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

Usa il AWS SDK for Rust

Scopri i modi più comuni e consigliati di utilizzarli AWS SDK for Rust per lavorare con AWS i servizi.

Argomenti

- [Fai richieste](#)
- [Best practice](#)
- [Simultaneità](#)
- [Gestione degli errori](#)
- [Creare funzioni Lambda](#)
- [Paginazione](#)
- [Crea predefiniti URLs](#)
- [Test unitari](#)
- [Waiter](#)

Fai richieste

Per fare una richiesta a un Servizio AWS, devi prima [creare un client di servizio](#). Per ogni Servizio AWS codice utilizzato, ha una propria cassa e una propria `Client` per interagire con esso.

`Client` Espone un metodo per ogni operazione API esposta dal servizio. Il valore restituito da ciascuno di questi metodi è un «generatore fluente», in cui diversi input per quell'API vengono aggiunti mediante il concatenamento di chiamate di funzioni in stile builder. Dopo aver chiamato i metodi del servizio, chiama `send()` per ottenere un risultato [Future](#) che genererà un output corretto o un `SdkError`. Per ulteriori informazioni su `SdkError`, consulta [Gestione degli errori](#).

L'esempio seguente illustra un'operazione di base con Amazon S3 per creare un bucket in: us-west-2 Regione AWS

```
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;
```

Ogni service crate dispone di moduli aggiuntivi utilizzati per gli input delle API, come i seguenti:

- Il `types` modulo dispone di strutture o enumerazioni per fornire informazioni strutturate più complesse.
- Il `primitives` modulo ha tipi più semplici per rappresentare dati come date, ore o blob binari.

Consulta la [documentazione di riferimento dell'API](#) per il service crate per informazioni e organizzazione più dettagliate delle casse. Ad esempio, la `aws-sdk-s3` cassa per Amazon Simple Storage Service ha diversi [moduli](#). Due dei quali sono:

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

Best practice

Di seguito sono riportate le migliori pratiche per l'utilizzo di AWS SDK for Rust

Riutilizza i client SDK quando possibile

A seconda di come viene costruito un client SDK, la creazione di un nuovo client può comportare che ogni client mantenga i propri pool di connessioni HTTP, cache di identità e così via. Ti consigliamo di condividere un client o almeno di dividerlo `SdkConfig` per evitare il sovraccarico legato alla creazione di costose risorse. Tutti i client SDK vengono implementati `Clone` come un unico aggiornamento del conteggio dei riferimenti atomici.

Configura i timeout delle API

L'SDK fornisce valori predefiniti per alcune opzioni di timeout, come il timeout di connessione e i timeout dei socket, ma non per i timeout delle chiamate API o i singoli tentativi di chiamata API. È buona norma impostare i timeout sia per il singolo tentativo che per l'intera richiesta. Ciò garantirà un guasto rapido dell'applicazione in modo ottimale in caso di problemi temporanei che potrebbero causare tempi più lunghi per il completamento dei tentativi di richiesta o problemi di rete irreversibili.

Per ulteriori informazioni sulla configurazione dei timeout operativi, vedere. [Timeout](#)

Simultaneità

AWS SDK for Rust Non fornisce il controllo della concorrenza, ma gli utenti hanno molte opzioni per implementarne di proprie.

Termini

I termini relativi a questo argomento sono facili da confondere e alcuni termini sono diventati sinonimi anche se originariamente rappresentavano concetti separati. In questa guida, definiremo quanto segue:

- **Attività:** Una «unità di lavoro» che il programma eseguirà fino al completamento o che tenterà di eseguire fino al completamento.
- **Calcolo sequenziale:** quando diverse attività vengono eseguite una dopo l'altra.
- **Calcolo simultaneo:** quando più attività vengono eseguite in periodi di tempo sovrapposti.
- **Concorrenza:** la capacità di un computer di completare più attività in un ordine arbitrario.
- **Multitasking:** la capacità di un computer di eseguire più attività contemporaneamente.
- **Condizione di gara:** quando il comportamento del programma cambia in base all'avvio di un'attività o al tempo necessario per elaborarla.
- **Controversia:** conflitto sull'accesso a una risorsa condivisa. Quando due o più attività vogliono accedere a una risorsa contemporaneamente, quella risorsa è «in lizza».
- **Deadlock:** uno stato in cui non è possibile fare ulteriori progressi. Ciò si verifica in genere perché due attività vogliono acquisire le rispettive risorse, ma nessuna delle due attività libererà la propria risorsa finché la risorsa dell'altra non sarà disponibile. I deadlock fanno sì che un programma non risponda parzialmente o completamente.

Un semplice esempio

Il nostro primo esempio è un programma sequenziale. Negli esempi successivi, cambieremo questo codice usando tecniche di concorrenza. Gli esempi successivi riutilizzano lo stesso `build_client_and_list_objects_to_download()` metodo e apportano modifiche all'interno. `main()`

L'attività di esempio seguente consiste nel scaricare tutti i file in un bucket Amazon Simple Storage Service:

1. Inizia elencando tutti i file. Salva le chiavi in un elenco.
2. Scorri l'elenco, scaricando ogni file a turno

```
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 questi esempi, non gestiremo gli errori e supponiamo che il bucket di esempio non contenga oggetti con chiavi che assomigliano a percorsi di file. Pertanto, non tratteremo la creazione di directory annidate.

Grazie all'architettura dei computer moderni, possiamo riscrivere questo programma per renderlo molto più efficiente. Lo faremo in un esempio successivo, ma prima impariamo qualche altro concetto.

Proprietà e mutabilità

Ogni valore in Rust ha un unico proprietario. Quando un proprietario esce dall'ambito di applicazione, verranno eliminati anche tutti i valori che possiede. Il proprietario può fornire uno o più riferimenti immutabili a un valore o un singolo riferimento mutabile. Il compilatore Rust è responsabile di garantire che nessun riferimento sopravviva al suo proprietario.

Sono necessarie una pianificazione e una progettazione aggiuntive quando più attività devono accedere in modo mutevole alla stessa risorsa. Nel calcolo sequenziale, ogni attività può accedere in modo mutabile alla stessa risorsa senza conflitti perché vengono eseguite una dopo l'altra in una sequenza. Tuttavia, nell'elaborazione concorrente, le attività possono essere eseguite in qualsiasi ordine e contemporaneamente. Pertanto, dobbiamo fare di più per dimostrare al compilatore che più riferimenti mutabili sono impossibili (o almeno che si bloccano se si verificano).

La libreria standard di Rust fornisce molti strumenti per aiutarci a raggiungere questo obiettivo. Per ulteriori informazioni su questi argomenti, consulta il libro [Variabili e mutabilità](#) e [comprensione della proprietà](#) nel linguaggio di programmazione Rust.

Altri termini!

Di seguito sono elencati gli «oggetti di sincronizzazione». Nel complesso, sono gli strumenti necessari per convincere il compilatore che il nostro programma concorrente non infrangerà le regole di proprietà.

Oggetti di sincronizzazione della [libreria standard](#):

- [Arco](#): un puntatore a riferimento atomico `R`, montato in `C`. Quando i dati sono racchiusi in un file, possono essere condivisi liberamente `Arc`, senza preoccuparsi che un proprietario specifico perda prematuramente il valore. In questo senso, la proprietà del valore diventa «condivisa». I valori all'interno di un `Arc` non possono essere mutevoli, ma potrebbero avere una [mutabilità interiore](#).
- [Barriera](#): assicura che più thread attendano che l'altro raggiunga un punto del programma, prima di continuare l'esecuzione tutti insieme.
- [Condvar](#): una variabile di condizione che consente di bloccare un thread in attesa che si verifichi un evento.
- [Mutex](#): un meccanismo di esclusione reciproca che assicura che al massimo un thread alla volta sia in grado di accedere ad alcuni dati. In generale, un `Mutex` lucchetto non dovrebbe mai essere posizionato su un `.await` punto del codice.

Oggetti di [sincronizzazione Tokio](#):

Sebbene AWS SDKs siano pensati per essere `async` indipendenti dal tempo di esecuzione, consigliamo l'uso di oggetti di sincronizzazione per casi specifici. `tokio`

- [Mutex](#): simile alle librerie standard `Mutex`, ma con un costo leggermente superiore. A differenza dello standard `Mutex`, questa può essere inserita in un `.await` punto del codice.
- [Semaphore](#): variabile utilizzata per controllare l'accesso a una risorsa comune mediante più attività.

Riscrivere il nostro esempio per renderlo più efficiente (concorrenza a thread singolo)

Nel seguente esempio modificato, eseguiamo TUTTE [futures_util::future::join_all](#) le richieste contemporaneamente. `get_object`

```
#[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;
}
```

Questo è il modo più semplice per trarre vantaggio dalla concorrenza, ma presenta anche alcuni problemi che potrebbero non essere evidenti a prima vista:

1. Creiamo tutti gli input di richiesta contemporaneamente. Se non abbiamo abbastanza memoria per contenere tutti gli input della `get_object` richiesta, ci imbatteremo in un errore di allocazione out-of-memory "».
2. Creiamo e aspettiamo tutti i futuri contemporaneamente. Amazon S3 limita le richieste se proviamo a scaricarne troppe alla volta.

Per risolvere entrambi questi problemi, dobbiamo limitare la quantità di richieste che inviamo contemporaneamente. Lo faremo con un tokio [semaforo](#):

```
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;
}
```

Abbiamo risolto il potenziale problema di utilizzo della memoria spostando la creazione della richiesta nel `async` blocco. In questo modo, le richieste non verranno create finché non sarà il momento di inviarle.

Note

Se disponi della memoria necessaria, potrebbe essere più efficiente creare tutti gli input di richiesta contemporaneamente e conservarli in memoria finché non sono pronti per essere inviati. Per provare ciò, sposta la creazione dell'input della richiesta all'esterno del `async` blocco.

Abbiamo anche risolto il problema dell'invio di troppe richieste contemporaneamente limitando le richieste in corso a `CONCURRENCY_LIMIT`.

Note

Il valore giusto per `CONCURRENCY_LIMIT` è diverso per ogni progetto. Quando crei e invii le tue richieste, prova a impostarle il più in alto possibile senza incorrere in errori di limitazione. Sebbene sia possibile aggiornare dinamicamente il limite di concorrenza in base al rapporto tra risposte riuscite e risposte limitate che un servizio restituisce, ciò non rientra nell'ambito di questa guida a causa della sua complessità.

Riscrivere il nostro esempio per renderlo più efficiente (concorrenza multithread)

Nei due esempi precedenti, abbiamo eseguito le nostre richieste contemporaneamente. Sebbene ciò sia più efficiente rispetto all'esecuzione sincrona, possiamo rendere le cose ancora più efficienti utilizzando il multithreading. Per farlo con `tokio`, dovremo generarli come attività separate.

Note

Questo esempio richiede l'utilizzo del runtime `multithreadtokio`. Questo runtime è protetto dalla funzionalità `rt-multi-thread`. E, ovviamente, dovrai eseguire il programma su una macchina multi-core.

```
// 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;
}
```

Dividere il lavoro in attività può essere complesso. Le operazioni di I/O (input/output) sono in genere bloccanti. I runtime potrebbero avere difficoltà a bilanciare le esigenze delle attività di lunga durata con quelle delle attività di breve durata. Qualunque sia il tempo di esecuzione che scegliete, assicuratevi di leggere i loro consigli per suddividere il lavoro in attività nel modo più efficiente possibile. Per i consigli tokio di runtime, consulta [Module tokio::task](#).

Eseguire il debug di app multithread

Le attività eseguite contemporaneamente possono essere eseguite in qualsiasi ordine. Pertanto, i registri dei programmi concorrenti possono essere molto difficili da leggere. Nell'SDK per Rust, consigliamo di utilizzare il `tracing` sistema di registrazione. Può raggruppare i log con le loro attività specifiche, indipendentemente da quando sono in esecuzione. Per le linee guida, consulta [Abilita la registrazione del codice AWS SDK for Rust](#).

Uno strumento molto utile per identificare le attività bloccate è [tokio-console](#) uno strumento di diagnostica e debug per programmi Rust asincroni. Strumentando ed eseguendo il programma, e quindi eseguendo `'tokio-console' app`, è possibile vedere una visualizzazione in tempo reale delle attività eseguite dal programma. Questa visualizzazione include informazioni utili come la quantità di tempo che un'attività ha impiegato in attesa di acquisire risorse condivise o la quantità di volte in cui è stata interrogata.

Gestione degli errori

Comprendere come e quando si verificano gli errori di AWS SDK for Rust restituzione è importante per creare applicazioni di alta qualità utilizzando l'SDK. Le sezioni seguenti descrivono i diversi errori che potreste riscontrare nell'SDK e come gestirli in modo appropriato.

Ogni operazione restituisce un `Result` tipo con il tipo di errore impostato su [SdkError<E, R = HttpResponse>](#). `SdkError` è un enum con diversi tipi possibili, chiamati varianti.

Errori del servizio

Il tipo di errore più comune è [SdkError::ServiceError](#). Questo errore rappresenta una risposta di errore da un Servizio AWS. Ad esempio, se cerchi di ottenere un oggetto da Amazon S3 che non esiste, Amazon S3 restituisce una risposta di errore.

Quando si verifica un messaggio `SdkError::ServiceError` significa che la richiesta è stata inviata correttamente a Servizio AWS ma non può essere elaborata. Questo può essere dovuto a errori nei parametri della richiesta o a errori sul lato servizio.

I dettagli della risposta all'errore sono inclusi nella variante di errore. L'esempio seguente mostra come accedere comodamente alla `ServiceError` variante sottostante e gestire diversi casi di errore:

```
// 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())
    }
};

```

Metadati di errore

Ogni errore di servizio contiene metadati aggiuntivi a cui è possibile accedere importando caratteristiche specifiche del servizio.

- La `<service>::error::ProvideErrorMetadata` caratteristica fornisce l'accesso a qualsiasi codice di errore non elaborato sottostante e messaggio di errore restituito dal servizio.
 - Per Amazon S3, questa caratteristica è [aws_sdk_s3::error::ProvideErrorMetadata](#)

Puoi anche ottenere informazioni che potrebbero essere utili per la risoluzione degli errori del servizio:

- La `<service>::operation::RequestId` caratteristica aggiunge metodi di estensione per recuperare l'ID univoco della AWS richiesta generato dal servizio.
 - Per Amazon S3, questa caratteristica è [aws_sdk_s3::operation::RequestId](#)
- La `<service>::operation::RequestIdExt` caratteristica aggiunge il `extended_request_id()` metodo per ottenere un ID di richiesta esteso aggiuntivo.

- Supportato solo da alcuni servizi.
- Per Amazon S3, questa caratteristica è [aws_sdk_s3::operation::RequestIdExt](#)

Errore di stampa dettagliato con **DisplayErrorContext**

Gli errori nell'SDK sono generalmente il risultato di una catena di errori come:

1. L'invio di una richiesta non è riuscito perché il connettore ha restituito un errore.
2. Il connettore ha restituito un errore perché il provider delle credenziali ha restituito un errore.
3. Il provider di credenziali ha restituito un errore perché ha chiamato un servizio e quel servizio ha restituito un errore.
4. Il servizio ha restituito un errore perché la richiesta di credenziali non aveva l'autorizzazione corretta.

Per impostazione predefinita, la visualizzazione di questo errore genera solo «errore di invio». Mancano dettagli che aiutino a risolvere l'errore. L'SDK per Rust fornisce un semplice segnalatore di errori chiamato `DisplayErrorContext`

- La `<service>::error::DisplayErrorContext` struttura aggiunge funzionalità per generare il contesto di errore completo.
- Per Amazon S3, questa struttura è [aws_sdk_s3::error::DisplayErrorContext](#)

Quando impacchettiamo l'errore da visualizzare e lo stampiamo, `DisplayErrorContext` fornisce un messaggio molto più dettagliato simile al seguente:

```
dispatch failure: other: Session token not found or invalid.
DispatchFailure(
  DispatchFailure {
    source: ConnectorError {
      kind: Other(None),
      source: ProviderError(
        ProviderError {
          source: ProviderError(
            ProviderError {
              source: ServiceError(
                ServiceError {
                  source: UnauthorizedException(
```



```

        UnauthorizedException {
            message: Some("Session token not found or
invalid"),
            meta: ErrorMetadata {
                code: Some("UnauthorizedException"),
                message: Some("Session token not found
or invalid"),
                extras: Some({"aws_request_id":
"1b6d7476-f5ec-4a16-9890-7684ccee7d01"})
            }
        },
        raw: Response {
            status: StatusCode(401),
            headers: Headers {
                headers: {
                    "date": HeaderValue { _private:
H0("Thu, 04 Jul 2024 07:41:21 GMT") },
                    "content-type": HeaderValue { _private:
H0("application/json") },
                    "content-length": HeaderValue
{ _private: H0("114") },
                    "access-control-expose-headers":
HeaderValue { _private: H0("RequestId") },
                    "access-control-expose-headers":
HeaderValue { _private: H0("x-amzn-RequestId") },
                    "requestid": HeaderValue { _private:
H0("1b6d7476-f5ec-4a16-9890-7684ccee7d01") },
                    "server": HeaderValue { _private:
H0("AWS SSO") },
                    "x-amzn-requestid": HeaderValue
{ _private: H0("1b6d7476-f5ec-4a16-9890-7684ccee7d01") }
                }
            },
            body: SdkBody {
                inner: Once(
                    Some(
                        b"{
                            \"message\": \"Session token not
found or invalid\",
                            \"__type\":
\"com.amazonaws.switchboard.portal#UnauthorizedException\"}"}
                    )
                ),

```

```
        retryable: true
    },
    extensions: Extensions {
        extensions_02x: Extensions,
        extensions_1x: Extensions
    }
}
)
),
connection: Unknown
}
}
)
```

Creare funzioni Lambda

Per una documentazione dettagliata sullo sviluppo di AWS Lambda funzioni con AWS SDK for Rust, consulta [Building Lambda functions with Rust](#) nella AWS Lambda Developer Guide. Questa documentazione ti guida nell'utilizzo di:

- La cassa del client di runtime Rust Lambda per le funzionalità di base, [aws-lambda-rust-runtime](#)
- [Lo strumento da riga di comando consigliato per distribuire il binario della funzione Rust su Lambda con Cargo Lambda.](#)

Oltre agli esempi guidati contenuti nella AWS Lambda Developer Guide, sono disponibili anche esempi di calcolatrice Lambda nell'[AWS SDK Code Examples Repository](#) su GitHub

Paginazione

Molte AWS operazioni restituiscono risultati troncati quando il payload è troppo grande per essere restituito in un'unica risposta. Il servizio restituisce invece una parte dei dati e un token per recuperare il set successivo di elementi. Questo modello è noto come impaginazione.

AWS SDK for Rust Include metodi di estensione `into_paginator` sugli Operation Builder che possono essere utilizzati per impaginare automaticamente i risultati. Devi solo scrivere il codice che elabora i risultati. Tutti i generatori di operazioni di impaginazione dispongono di un `into_paginator()` metodo che impone a di [PaginationStream<Item>](#) impaginare i risultati.

- In Amazon S3, un esempio di ciò è.

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

I seguenti esempi utilizzano Amazon Simple Storage Service. Tuttavia, i concetti sono gli stessi per qualsiasi servizio che ha una o più pagine APIs.

Il seguente esempio di codice mostra l'esempio più semplice che utilizza il [try_collect\(\)](#) metodo per raccogliere tutti i risultati impaginati in un file: `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<_>>();
```

A volte, si desidera avere un maggiore controllo sul paging e non archiviare tutto in memoria contemporaneamente. L'esempio seguente esegue iterazioni sugli oggetti in un bucket Amazon S3 finché non ce ne sono più.

```
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);
    }
}
```

Crea predefiniti URLs

È possibile preassegnare le richieste per alcune operazioni AWS API in modo che un altro chiamante possa utilizzare la richiesta in un secondo momento senza presentare le proprie credenziali.

Ad esempio, supponiamo che Jane abbia accesso a un oggetto Amazon Simple Storage Service (Amazon S3) e desideri condividere temporaneamente l'accesso agli oggetti con Alejandro. Jane può generare una `GetObject` richiesta predefinita da condividere con Alejandro in modo che quest'ultimo possa scaricare l'oggetto senza dover accedere alle credenziali di Jane o averne di proprie. Le credenziali utilizzate dall'URL predefinito sono di Jane perché è l'utente che ha generato l'URL. AWS

Per ulteriori informazioni sulle impostazioni predefinite URLs in Amazon S3, [consulta Working with URLs](#) presigned nella Amazon Simple Storage Service User Guide.

Nozioni di base sulla preassegnazione

AWS SDK for Rust Fornisce un `presigned()` metodo sul funzionamento fluent-builders che può essere utilizzato per ottenere una richiesta prefirmata.

L'esempio seguente crea una `GetObject` richiesta predefinita per Amazon S3. La richiesta è valida per 5 minuti dopo la creazione.

```
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?;

```

Il `presigned()` metodo restituisce un `Result<PresignedRequest, SdkError<E, R>>`.

Il valore restituito `PresignedRequest` contiene metodi per accedere ai componenti di una richiesta HTTP, inclusi il metodo, l'URI ed eventuali intestazioni. Tutti questi devono essere inviati al servizio, se presente, affinché la richiesta sia valida. Tuttavia, molte richieste predefinite possono essere rappresentate solo dall'URI.

Preassegnazione **POST** e richieste **PUT**

Molte operazioni prefirmabili richiedono solo un URL e devono essere inviate come richieste HTTP. GET Alcune operazioni, tuttavia, richiedono un corpo e in alcuni casi devono essere inviate come PUT richiesta HTTP POST o HTTP insieme alle intestazioni. La preassegnazione di queste richieste è identica alla prefirma GET delle richieste, ma richiamare la richiesta prefirmata è più complicata.

Di seguito è riportato un esempio di preassegnazione di una richiesta Amazon PutObject S3 e la conversione in [http::request::Request](#) una richiesta che può essere inviata utilizzando un client HTTP di tua scelta.

Per utilizzare il `into_http_1x_request()` metodo, aggiungi la `http-1x` funzionalità alla casella del `aws-sdk-s3` file: `Cargo.toml`

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

File sorgente:

```
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);
```

Firmatario autonomo

Note

Si tratta di un caso d'uso avanzato. Non è necessario o consigliato per la maggior parte degli utenti.

Esistono alcuni casi d'uso in cui è necessario creare una richiesta firmata al di fuori del contesto SDK for Rust. Per questo puoi usare la [aws-sigv4](#) cassa indipendentemente dall'SDK.

Quello che segue è un esempio per dimostrare gli elementi di base, consulta la documentazione del crate per maggiori dettagli.

Aggiungi le http casse `aws-sigv4` e al tuo `Cargo.toml` file:

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

File sorgente:

```
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);
```

Test unitari

Sebbene ci siano molti modi per implementare i test unitari nel tuo AWS SDK for Rust progetto, ce ne sono alcuni che consigliamo:

- Utilizzalo [automock](#) from the [mockall](#) crate per creare ed eseguire i tuoi test.
- Usa il runtime di AWS Smithy [StaticReplayClient](#) per creare un falso client HTTP che può essere usato al posto del client HTTP standard che viene normalmente utilizzato da. Servizi AWS

Questo client restituisce le risposte HTTP specificate anziché comunicare con il servizio tramite la rete, in modo che i test ottengano dati noti a scopo di test.

Genera automaticamente dei mock usando mockall

Puoi generare automaticamente la maggior parte delle implementazioni fittizie di cui i tuoi test hanno bisogno utilizzando il popolare programma `from the crate`. [automock mockall](#)

Questo esempio verifica un metodo personalizzato chiamato `determine_prefix_file_size()`. Questo metodo chiama un metodo `list_objects()` wrapper personalizzato che chiama Amazon S3. Prendendo in giro `list_objects()`, il `determine_prefix_file_size()` metodo può essere testato senza contattare effettivamente Amazon S3.

1. In un prompt dei comandi per la directory del tuo progetto, aggiungi la [mockall](#) cassa come dipendenza:

```
$ cargo add mockall
```

Questo aggiunge la cassa alla `[dependencies]` sezione del file `Cargo.toml`

2. Includi il `automock` modulo dalla `mockall` cassa.

Includi anche qualsiasi altra libreria correlata a Servizio AWS quella che stai testando, in questo caso Amazon S3.

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

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

3. Quindi, aggiungi il codice che determina quale delle due implementazioni della struttura wrapper Amazon S3 dell'applicazione utilizzare.
 - Quella vera scritta per accedere ad Amazon S3 tramite la rete.
 - L'implementazione fittizia generata da `mockall`

In questo esempio, a quella selezionata viene assegnato il nome `S3`. La selezione è condizionata in base all'attributo `test`:


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

4. La `S3Impl` struttura è l'implementazione della struttura wrapper di Amazon S3 a cui invia effettivamente le richieste. AWS
- Quando il test è abilitato, questo codice non viene utilizzato perché la richiesta viene inviata al mock e non. AWS L'`dead_code` attributo dice al linter di non segnalare un problema se il `S3Impl` tipo non viene utilizzato.
 - Il condizionale `#[cfg_attr(test, automock)]` indica che quando il test è abilitato, l'`automock` attributo deve essere impostato. Questo dice `mockall` di generare un mock a `S3Impl` cui verrà dato un nome. `MockS3Impl`
 - In questo esempio, il `list_objects()` metodo è la chiamata che vuoi desiderare. `automock` creerà automaticamente un `expect_list_objects()` metodo per te.

```
#[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. Crea le funzioni di test in un modulo denominato `test`.

- Il condizionale `#[cfg(test)]` indica che `mockall` dovrebbe creare il modulo di test se l'attributo `test` è `true`.

```

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

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

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

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

#[tokio::test]

```

```

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

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

    assert_eq!(19, size);
}
}

```

- Ogni test utilizza `let mut mock = MockS3Impl::default();` per creare un'istanza di `MockS3Impl`.

- Utilizza il `expect_list_objects()` metodo mock's (che è stato creato automaticamente da `automock`) per impostare il risultato previsto per quando il `list_objects()` metodo viene utilizzato altrove nel codice.
 - Dopo aver stabilito le aspettative, le utilizza per testare la funzione `determine_prefix_file_size()` chiamando. Il valore restituito viene controllato per confermare che sia corretto, utilizzando un'asserzione.
6. La `determine_prefix_file_size()` funzione utilizza il wrapper Amazon S3 per ottenere la dimensione del file del prefisso:

```
#[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)
}
```

Il tipo `S3` viene utilizzato per chiamare le funzioni `Wrapped SDK for Rust` per supportare entrambe `S3Impl` le funzioni e per effettuare richieste HTTP. `MockS3Impl` Il mock generato automaticamente da `mockall` segnala eventuali errori di test quando il test è abilitato.

È possibile [visualizzare il codice completo per questi esempi](#) su GitHub

Simula il traffico HTTP utilizzando la riproduzione statica

La `aws-smithy-runtime` cassa include una classe di utilità di test chiamata.

[StaticReplayClient](#) Questa classe client HTTP può essere specificata al posto del client HTTP predefinito durante la creazione di un Servizio AWS oggetto.

Quando si inizializza `StaticReplayClient`, si fornisce un elenco di coppie di richieste e risposte HTTP come `ReplayEvent` oggetti. Durante l'esecuzione del test, ogni richiesta HTTP viene registrata e il client restituisce la risposta HTTP successiva trovata nella successiva `ReplayEvent` nell'elenco degli eventi come risposta del client HTTP. Ciò consente l'esecuzione del test utilizzando dati noti e senza una connessione di rete.

Utilizzo della riproduzione statica

Per utilizzare la riproduzione statica, non è necessario utilizzare un wrapper. Invece, stabilisci come dovrebbe essere l'effettivo traffico di rete per i dati che verranno utilizzati dal test e fornisci tali dati sul traffico all'utente da utilizzare ogni volta che l'SDK invia una richiesta dal client.

`StaticReplayClient` Servizio AWS

Note

Esistono diversi modi per raccogliere il traffico di rete previsto, tra cui numerosi analizzatori del traffico di rete AWS CLI e strumenti di analisi dei pacchetti.

- Crea un elenco di `ReplayEvent` oggetti che specificano le richieste HTTP previste e le risposte che devono essere restituite per esse.
- Crea un elenco di transazioni `StaticReplayClient` utilizzando l'elenco di transazioni HTTP creato nel passaggio precedente.
- Crea un oggetto di configurazione per il AWS client, specificandolo `StaticReplayClient` come `Config` oggetto. `http_client`
- Crea l'oggetto Servizio AWS client, utilizzando la configurazione creata nel passaggio precedente.

- Eseguite le operazioni che desiderate testare utilizzando l'oggetto servizio configurato per utilizzare il `StaticReplayClient`. Ogni volta che l'SDK invia una richiesta API a AWS, viene utilizzata la risposta successiva nell'elenco.

Note

La risposta successiva nell'elenco viene sempre restituita, anche se la richiesta inviata non corrisponde a quella nel vettore di `ReplayEvent` oggetti.

- Una volta effettuate tutte le richieste desiderate, chiamate la `StaticReplayClient.assert_requests_match()` funzione per verificare che le richieste inviate dall'SDK corrispondano a quelle nell'elenco degli `ReplayEvent` oggetti.

Esempio

Diamo un'occhiata ai test per la stessa `determine_prefix_file_size()` funzione nell'esempio precedente, ma utilizziamo la riproduzione statica anziché la simulazione.

1. In un prompt dei comandi per la directory del progetto, aggiungi la [aws-smithy-runtime](#) cassa come dipendenza:

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

Questo aggiunge la cassa alla `[dependencies]` sezione del file `Cargo.toml`

2. Nel file sorgente, includi i `aws_smithy_runtime` tipi di cui avrai bisogno.

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

3. Il test inizia creando le `ReplayEvent` strutture che rappresentano ciascuna delle transazioni HTTP che dovrebbero avvenire durante il test. Ogni evento contiene un oggetto di richiesta HTTP e un oggetto di risposta HTTP che rappresentano le informazioni con cui normalmente Servizio AWS risponderebbero. Questi eventi vengono passati in una chiamata a `StaticReplayClient::new()`:

```
let page_1 = ReplayEvent::new(
    http::Request::builder()
        .method("GET")
```

```

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

```

Il risultato viene memorizzato in `replay_client`. Rappresenta un client HTTP che può quindi essere utilizzato dall'SDK per Rust specificandolo nella configurazione del client.

4. Per creare il client Amazon S3, chiama la `from_conf()` funzione della classe `client` per creare il client utilizzando un oggetto di configurazione:

```

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(),
);

```

L'oggetto di configurazione viene specificato utilizzando il `http_client()` metodo del generatore e le credenziali vengono specificate utilizzando il metodo.

`credentials_provider()` Le credenziali vengono create utilizzando una funzione chiamata `make_s3_test_credentials()`, che restituisce una struttura di credenziali false:

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

Queste credenziali non devono essere valide perché non verranno effettivamente inviate a AWS.

5. Esegui il test chiamando la funzione che deve essere testata. In questo esempio, il nome di quella funzione è `determine_prefix_file_size()`. Il suo primo parametro è l'oggetto client Amazon S3 da utilizzare per le sue richieste. Pertanto, specifica il client creato utilizzando il sistema `StaticReplayClient` in modo che le richieste vengano gestite da quello anziché essere inviate in rete:

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

assert_eq!(19, size);

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

Al termine della chiamata a `determine_prefix_file_size()` viene utilizzata un'asserzione per confermare che il valore restituito corrisponde al valore previsto. Quindi, viene chiamata la `assert_requests_match()` funzione del `StaticReplayClient` metodo. Questa funzione analizza le richieste HTTP registrate e conferma che corrispondono tutte a quelle specificate nell'array di `ReplayEvent` oggetti fornito durante la creazione del client di replay.

È possibile [visualizzare il codice completo per questi esempi](#) su GitHub

Waiter

I camerieri sono un'astrazione lato client utilizzata per interrogare una risorsa fino al raggiungimento dello stato desiderato o fino a quando non viene stabilito che la risorsa non entrerà nello stato desiderato. Questa è un'attività comune quando si lavora con servizi che alla fine sono coerenti, come Amazon Simple Storage Service, o servizi che creano risorse in modo asincrono, come Amazon Elastic Compute Cloud. Scrivere una logica per controllare continuamente lo stato di una risorsa può essere complicato e soggetto a errori. L'obiettivo dei camerieri è trasferire questa responsabilità dal codice del cliente alla società AWS SDK for Rust, che ha una conoscenza approfondita degli aspetti relativi alle tempistiche dell'operazione. AWS

Servizi AWS che forniscono supporto ai camerieri includono un modulo. `<service>::waiters`

- La `<service>::client::Waiters` caratteristica fornisce metodi di cameriere per il cliente. I metodi sono implementati per la `Client` struttura. Tutti i metodi di cameriere seguono una convenzione di denominazione standard di `wait_until_<Condition>`
 - Per Amazon S3, questa caratteristica è [aws_sdk_s3::client::Waiters](#)

L'esempio seguente utilizza Amazon S3. Tuttavia, i concetti sono gli stessi per tutti quelli Servizio AWS che hanno uno o più camerieri definiti.

Il seguente esempio di codice mostra l'utilizzo di una funzione di cameriere invece di scrivere una logica di polling per attendere l'esistenza di un bucket dopo essere stato creato.

```
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

Ogni metodo di attesa restituisce una risposta `Result<FinalPoll<...>, WaiterError<...>>` che può essere utilizzata per ottenere la risposta finale dopo il raggiungimento della condizione desiderata o di un errore. Vedi [FinalPoll](#) e [WaiterError](#) nella documentazione dell'API Rust per i dettagli.

Esempi di codice SDK per Rust

Gli esempi di codice in questo argomento mostrano come utilizzare l' AWS SDK per Rust con. AWS

Le nozioni di base sono esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Gli scenari sono esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Alcuni servizi contengono categorie di esempio aggiuntive che mostrano come sfruttare le librerie o le funzioni specifiche del servizio.

Servizi

- [Esempi di API Gateway che utilizzano SDK per Rust](#)
- [Esempi di API Gateway Management con SDK per Rust](#)
- [Esempi di Application Auto Scaling con SDK for Rust](#)
- [Esempi di Aurora con SDK per Rust](#)
- [Esempi di Auto Scaling con SDK per Rust](#)
- [Esempi di Amazon Bedrock Runtime con SDK per Rust](#)
- [Esempi di Amazon Cognito Identity Provider che utilizzano SDK per Rust](#)
- [Esempi di Amazon Cognito Sync con SDK per Rust](#)
- [Esempi di Firehose che utilizzano SDK per Rust](#)
- [Esempi di Amazon DocumentDB con SDK per Rust](#)
- [Esempi di DynamoDB con SDK per Rust](#)
- [Esempi di Amazon EBS con SDK per Rust](#)
- [EC2 Esempi di Amazon che utilizzano SDK per Rust](#)
- [Esempi di Amazon ECR con SDK per Rust](#)
- [Esempi di Amazon ECS con SDK per Rust](#)

- [Esempi di Amazon EKS che utilizzano SDK per Rust](#)
- [AWS Glue esempi che utilizzano SDK per Rust](#)
- [Esempi di IAM che utilizzano SDK per Rust](#)
- [AWS IoT esempi che utilizzano SDK per Rust](#)
- [Esempi di Kinesis con SDK per Rust](#)
- [AWS KMS esempi che utilizzano SDK per Rust](#)
- [Esempi di Lambda con SDK per Rust](#)
- [MediaLive esempi che utilizzano SDK per Rust](#)
- [MediaPackage esempi che utilizzano SDK per Rust](#)
- [Esempi di Amazon MSK con SDK per Rust](#)
- [Esempi di Amazon Polly con SDK per Rust](#)
- [Esempi di QLDB che utilizzano SDK per Rust](#)
- [Esempi di Amazon RDS con SDK per Rust](#)
- [Esempi di Amazon RDS Data Service con SDK per Rust](#)
- [Esempi di Amazon Rekognition con SDK per Rust](#)
- [Esempi di Route 53 che utilizzano SDK per Rust](#)
- [Esempi di Amazon S3 con SDK per Rust](#)
- [SageMaker Esempi di intelligenza artificiale che utilizzano SDK per Rust](#)
- [Esempi di Secrets Manager con SDK per Rust](#)
- [Esempi di API Amazon SES v2 con SDK per Rust](#)
- [Esempi di Amazon SNS con SDK per Rust](#)
- [Esempi di Amazon SQS con SDK per Rust](#)
- [AWS STS esempi che utilizzano SDK per Rust](#)
- [Esempi di Systems Manager che utilizzano SDK per Rust](#)
- [Esempi di Amazon Transcribe con SDK per Rust](#)

Esempi di API Gateway che utilizzano SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l' AWS SDK per Rust con API Gateway.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Gli scenari sono esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

AWS i contributi della community sono esempi che sono stati creati e gestiti da più team AWS. Per fornire feedback, utilizza il meccanismo fornito negli archivi collegati.

Ogni esempio include un collegamento al codice sorgente completo, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Operazioni](#)
- [Scenari](#)
- [AWS contributi della comunità](#)

Operazioni

GetRestApis

Il seguente esempio di codice mostra come utilizzare `GetRestApis`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Visualizza il REST di Amazon API Gateway APIs nella regione.

```
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(())
}
```

- Per i dettagli sull'API, consulta il riferimento [GetRestApis](#) all'API AWS SDK for Rust.

Scenari

Creazione di un'applicazione serverless per gestire foto

Nell'esempio di codice seguente viene illustrato come creare un'applicazione serverless che consente agli utenti di gestire le foto mediante etichette.

SDK per Rust

Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su [GitHub](#).

Per approfondire l'origine di questo esempio, consulta il post su [AWS Community](#).

Servizi utilizzati in questo esempio

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition

- Amazon S3
- Amazon SNS

AWS contributi della comunità

Crea e testa un'applicazione serverless

Il seguente esempio di codice mostra come creare e testare un'applicazione serverless utilizzando API Gateway con Lambda e DynamoDB.

SDK per Rust

Mostra come creare e testare un'applicazione serverless composta da un API Gateway con Lambda e DynamoDB utilizzando Rust SDK.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. [GitHub](#)

Servizi utilizzati in questo esempio

- API Gateway
- DynamoDB
- Lambda

Esempi di API Gateway Management con SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l' AWS SDK per Rust con API Gateway Management API.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Operazioni](#)

Operazioni

PostToConnection

Il seguente esempio di codice mostra come utilizzare `PostToConnection`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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);
```


- Per i dettagli sulle API, consulta il riferimento [PostToConnection](#) all'API AWS SDK for Rust.

Esempi di Application Auto Scaling con SDK for Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l'AWS SDK per Rust con Application Auto Scaling.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Operazioni](#)

Operazioni

DescribeScalingPolicies

Il seguente esempio di codice mostra come utilizzare `DescribeScalingPolicies`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

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

Ok(())
}
```

- Per i dettagli sulle API, consulta il riferimento [DescribeScalingPolicies](#) all'API AWS SDK for Rust.

Esempi di Aurora con SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l' AWS SDK per Rust con Aurora.

Le nozioni di base sono esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Hello Aurora

Gli esempi di codice seguenti mostrano come iniziare a utilizzare Aurora.

SDK per Rust

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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_idenfier().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(())
}

```

- Per i dettagli sull'API, consulta [Descrivi DBClusters](#) nella AWS guida di riferimento all'API SDK for Rust.

Argomenti

- [Nozioni di base](#)
- [Operazioni](#)

Nozioni di base

Informazioni di base

L'esempio di codice seguente mostra come:

- Crea un gruppo di parametri del cluster di database Aurora personalizzati e imposta i relativi valori.
- Crea un cluster di database che utilizza il gruppo di parametri.
- Crea un'istanza database che contiene un database.
- Acquisisci uno snapshot del cluster di database, quindi elimina le risorse.

SDK per Rust

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Una libreria contenente le funzioni specifiche per lo scenario Aurora.

```
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-mysql15.7}.
    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;

```

Test per la libreria attraverso automock per il wrapper del client RDS.

```

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_id(cluster)
                    .db_instance_id(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_id(id).build())
            .build())
    });

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

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



```

        Ok(DescribeDbClusterEndpointsOutput::builder()

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

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

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

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

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

```

```

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

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

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

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

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

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

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

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

    mock_rds
        .expect_create_db_cluster()

```

```

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

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

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

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

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

    mock_rds
        .expect_create_db_cluster()

```

```

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

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

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

```

```

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

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

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

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

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

```

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

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

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

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

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

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

```

        Ok(DescribeDbClustersOutput::builder()
            .db_clusters(
                DbCluster::builder()
                    .db_cluster_identifier(id)
                    .status("Deleting")
                    .build(),
            )
            .build())
    })
    .with(eq("MockCluster"))
    .times(1)
    .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

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

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

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

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

```

```
}

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

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

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

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

    mock_rds
        .expect_describe_db_clusters()
```



```

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

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

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

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

```

```

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

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

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

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

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

```

```

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

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

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

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

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

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

```

Un file binario per eseguire lo scenario dall'inizio alla fine, utilizzando un inquirer per consentire all'utente di prendere decisioni.

```

use std::fmt::Display;

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

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

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

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

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

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

```

```

}

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

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

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

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

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

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

```

```

    }

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

    Ok(scenario)
}

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

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

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

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

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

```

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

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

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

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

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

    Ok(())
}
```

```

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

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

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

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

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

    Ok(())
}

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

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

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

```



```

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

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

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

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

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

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

```

```

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

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

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

```

Un wrapper per il servizio Amazon RDS che consente l'automocking per i test.

```

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_idenfifier(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_idenfifier(cluster_name)
        .db_instance_idenfifier(instance_name)
        .db_instance_class(instance_class)
        .engine(engine)
        .send()
}
```

```
        .await
    }

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

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

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

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

```
}

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

pub async fn delete_db_cluster(
    &self,
    cluster_identifier: &str,
) -> Result<DeleteDbClusterOutput, SdkError<DeleteDBClusterError>> {
    self.inner
        .delete_db_cluster()
        .db_cluster_identifier(cluster_identifier)
        .skip_final_snapshot(true)
        .send()
        .await
}

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

Il file Cargo.toml con dipendenze utilizzato in questo scenario.


```
[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"] }
```

- Per informazioni dettagliate sulle API, consulta i seguenti argomenti nella Documentazione di riferimento delle API SDK AWS per Rust.
 - [CreaDBCluster](#)
 - [CreaDBClusterParameterGroup](#)
 - [Crea DBCluster istantanea](#)
 - [CreaDBInstance](#)
 - [EliminaDBCluster](#)
 - [EliminaDBClusterParameterGroup](#)
 - [EliminaDBInstance](#)
 - [Descriva DBCluster ParameterGroups](#)
 - [DBClusterDescrivi parametri](#)

- [Descrivi le DBCluster istantanee](#)
- [Descriva DBClusters](#)
- [Descrivi DBEngine versioni](#)
- [Descriva DBInstances](#)
- [DescribeOrderableDBInstanceOpzioni](#)
- [ModificaDBClusterParameterGroup](#)

Operazioni

CreateDBCluster

Il seguente esempio di codice mostra come usare `CreateDBCluster`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
// 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_identifer = create_db_instance
    .unwrap()
    .db_instance
    .and_then(|i| i.db_instance_identifer);

// 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_identifer
                .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_identifer
                .as_deref()
                .expect("instance identifier"),
        )
        .await;
    if let Err(err) = instance {
        return Err(ScenarioError::new(
```

```
        "Failed to find instance for cluster",
        &err,
    ));
}

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

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

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

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

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

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

pub async fn create_db_cluster(
    &self,
    name: &str,
```

```

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

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

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

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

```

```

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

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

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

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

```

```

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

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

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

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

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

```



```

        ))
    });

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

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

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

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

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

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

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

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {

```

```

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

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

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

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

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

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {

```

```

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

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

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

```

```

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

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

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

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

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

```

```

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

```

- Per i dettagli sull'API, consulta [DBClusterCreate](#) in AWS SDK for Rust API reference.

CreateDBClusterParameterGroup

Il seguente esempio di codice mostra come utilizzare `CreateDBClusterParameterGroup`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

// 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());
}

```

- Per i dettagli sull'API, consulta [DBClusterParameterGroupCreate](#) in AWS SDK for Rust API reference.

CreateDBClusterSnapshot

Il seguente esempio di codice mostra come utilizzare `CreateDBClusterSnapshot`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

// 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;
}

```

- Per i dettagli sull'API, consulta [Create DBCluster Snapshot](#) in AWS SDK for Rust API reference.

CreateDBInstance

Il seguente esempio di codice mostra come utilizzare `CreateDBInstance`

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

// 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_idenfier(cluster)
                        .db_instance_idenfier(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_idenfier(id).build())
            .build())
    });

mock_rds
    .expect_describe_db_instance()
    .with(eq("RustSDKCodeExamplesDBInstance"))
    .return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_idenfier(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;
}

```

- Per i dettagli sull'API, consulta [DBInstanceCreate](#) in AWS SDK for Rust API reference.

DeleteDBCluster

Il seguente esempio di codice mostra come utilizzare `DeleteDBCluster`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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

    // Delete the instance. rds.DeleteDbInstance.
    let delete_db_instance = self
        .rds
        .delete_db_instance(
            self.db_instance_identifier
                .as_deref()
                .expect("instance identifier"),
        )
        .await;
    if let Err(err) = delete_db_instance {
        let identifier = self
            .db_instance_identifier
            .as_deref()
            .unwrap_or("Missing Instance Identifier");
        let message = format!("failed to delete db instance {identifier}");
    }
}

```

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

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

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

```

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

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

```



```

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

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

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

pub async fn delete_db_cluster(
    &self,
    cluster_identifier: &str,
) -> Result<DeleteDbClusterOutput, SdkError<DeleteDBClusterError>> {
    self.inner
        .delete_db_cluster()
        .db_cluster_identifier(cluster_identifier)

```

```
        .skip_final_snapshot(true)
        .send()
        .await
    }

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

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

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

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

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

```

        DbCluster::builder()
            .db_cluster_identifier(id)
            .status("Deleting")
            .build(),
    )
    .build()
})
.with(eq("MockCluster"))
.times(1)
.returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

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

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

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

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

```

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

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

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

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

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
```

```

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

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

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

tokio::time::pause();
let assertions = tokio::spawn(async move {
    let clean_up = scenario.clean_up().await;
    assert!(clean_up.is_err());
    let errs = clean_up.unwrap_err();
    assert_eq!(errs.len(), 2);
});

```

```

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

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

```

- Per i dettagli sull'API, consulta [Delete DBCluster](#) in AWS SDK for Rust API reference.

DeleteDBClusterParameterGroup

Il seguente esempio di codice mostra come utilizzare `DeleteDBClusterParameterGroup`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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_identififier
                .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_idenfifier("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;
}

```

- Per i dettagli sull'API, consulta [Delete DBCluster ParameterGroup](#) in AWS SDK for Rust API reference.

DeleteDBInstance

Il seguente esempio di codice mostra come utilizzare `DeleteDBInstance`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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

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

            if db_instances.is_empty() {
                trace!("Delete Instance waited and no instances were found");
                break;
            }
        }
    }
}

```

```

    }
    match db_instances.first().unwrap().db_instance_status() {
        Some("Deleting") => continue,
        Some(status) => {
            info!("Attempting to delete but instances is in {status}");
            continue;
        }
        None => {
            warn!("No status for DB instance");
            break;
        }
    }
}
}

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

if let Err(err) = delete_db_cluster {
    let identifier = self
        .db_cluster_identifier
        .as_deref()
        .unwrap_or("Missing DB Cluster Identifier");
    let message = format!("failed to delete db cluster {identifier}");
    clean_up_errors.push(ScenarioError::new(message, &err));
} else {
    // Wait for the instance and cluster to fully delete.
    rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
    let waiter = Waiter::default();
    while waiter.sleep().await.is_ok() {
        let describe_db_clusters = self
            .rds
            .describe_db_clusters(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )

```



```

        .await;
    if let Err(err) = describe_db_clusters {
        clean_up_errors.push(ScenarioError::new(
            "Failed to check cluster state during deletion",
            &err,
        ));
        break;
    }
    let describe_db_clusters = describe_db_clusters.unwrap();
    let db_clusters = describe_db_clusters.db_clusters();
    if db_clusters.is_empty() {
        trace!("Delete cluster waited and no clusters were found");
        break;
    }
    match db_clusters.first().unwrap().status() {
        Some("Deleting") => continue,
        Some(status) => {
            info!("Attempting to delete but clusters is in {status}");
            continue;
        }
        None => {
            warn!("No status for DB cluster");
            break;
        }
    }
}

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

```

```

        "Failed to delete the db cluster parameter group",
        &error,
    ))
}

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

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

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

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

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

```

```

        )
        .build())
    })
    .with()
    .times(1)
    .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));

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

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

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

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

```

```

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

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

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

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

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

```

```

        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
            ))
        });

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

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

mock_rds
    .expect_delete_db_cluster_parameter_group()
    .with(eq("MockParamGroup"))

```

```

        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

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

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

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


```

- Per i dettagli sull'API, consulta [Delete DBInstance](#) in AWS SDK for Rust API reference.

DescribeDBClusterParameters

Il seguente esempio di codice mostra come utilizzare `DescribeDBClusterParameters`.

SDK per Rust

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
// 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");
}


```

- Per i dettagli sull'API, consulta [DBClusterDescrivi i parametri](#) nell'AWS SDK for Rust API reference.

DescribeDBClusters

Il seguente esempio di codice mostra come utilizzare `DescribeDBClusters`.

SDK per Rust

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

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

```
        return Err(ScenarioError::new(
            "Failed to create DB Cluster with cluster group",
            &err,
        ));
    }

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

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

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

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

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

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

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

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

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

    let endpoints = self
        .rds
        .describe_db_cluster_endpoints(
            self.db_cluster_identifier
                .as_deref()

```

```

        .expect("cluster identifier"),
    )
    .await;

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

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

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

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

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

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

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
        });
}

```

```

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

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

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

mock_rds

```

```

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

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

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

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

```

```

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

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

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

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

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

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

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

```



```

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

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

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

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

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

```

```

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

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

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

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

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

```

```

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

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

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

mock_rds
    .expect_describe_db_cluster_endpoints()
    .return_once(|_| {

```

```

        Ok(DescribeDbClusterEndpointsOutput::builder()

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

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

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

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

```

- Per i dettagli sull'API, consulta [Descrivi DBClusters](#) nella AWS guida di riferimento all'API SDK for Rust.

DescribeDBEngineVersions

Il seguente esempio di codice mostra come utilizzare `DescribeDBEngineVersions`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

    // 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"
    );
}

```

- Per i dettagli sull'API, consulta [Descrivi DBEngine le versioni](#) nell'AWS SDK per il riferimento all'API Rust.

DescribeDBInstances

Il seguente esempio di codice mostra come utilizzare `DescribeDBInstances`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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 identifiser = self
        .db_cluster_identifiser
        .as_deref()
        .unwrap_or("Missing DB Cluster Identifiser");
    let message = format!("failed to delete db cluster {identifiser}");
    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_identififier = 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;
}

```

- Per i dettagli sull'API, consulta [Descrivi DBInstances](#) nella AWS guida di riferimento all'API SDK for Rust.

DescribeOrderableDBInstanceOptions

Il seguente esempio di codice mostra come utilizzare `DescribeOrderableDBInstanceOptions`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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"
);
}

```

- Per i dettagli sull'API, consulta [DescribeOrderableDBInstanceOptions](#) in AWS SDK for Rust API reference.

ModifyDBClusterParameterGroup

Il seguente esempio di codice mostra come utilizzare `ModifyDBClusterParameterGroup`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

// 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");
}
```

- Per i dettagli sull'API, consulta [Modify DBCluster ParameterGroup](#) in AWS SDK for Rust API reference.

Esempi di Auto Scaling con SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l' AWS SDK per Rust con Auto Scaling.

Le nozioni di base sono esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Ciao Auto Scaling

I seguenti esempi di codice mostrano come iniziare a usare Auto Scaling.

SDK per Rust

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta il riferimento [DescribeAutoScalingGroups](#) all'API AWS SDK for Rust.

Argomenti

- [Nozioni di base](#)
- [Operazioni](#)

Nozioni di base

Informazioni di base

L'esempio di codice seguente mostra come:

- Crea un gruppo Amazon EC2 Auto Scaling con un modello di lancio e zone di disponibilità e ottieni informazioni sulle istanze in esecuzione.

- Abilita la raccolta di CloudWatch metriche Amazon.
- Aggiorna la capacità desiderata del gruppo e attendi l'avvio di un'istanza.
- Termina un'istanza nel gruppo.
- Elenca le attività di scalabilità che si verificano in risposta alle richieste degli utenti e ai cambiamenti di capacità.
- Ottieni statistiche per le CloudWatch metriche, quindi ripulisci le risorse.

SDK per Rust

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

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

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

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



```

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

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

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

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

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

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

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

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

    let mut warnings = Warnings::default();

    // 1. Create an EC2 launch template that you'll use to create an auto scaling
    group. Bonus: use SDK with EC2.CreateLaunchTemplate to create the launch template.

```

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

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

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

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

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

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

```
.await;

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

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

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

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

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

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

```

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

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

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

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

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

// 12. DeleteAutoScalingGroup (to delete the group you must stop all instances):
// 13. Delete LaunchTemplate.
let clean_scenario = scenario.clean_scenario().await;
if let Err(errs) = clean_scenario {
    for err in errs {

```

```

        warnings.push("There was a problem cleaning the scenario", err);
    }
} else {
    info!("The scenario has been cleaned up!");
}

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

pub mod scenario;

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

use anyhow::anyhow;
use aws_config::SdkConfig;
use aws_sdk_autoscaling::{
    error::{DisplayErrorContext, ProvideErrorMetadata},
    types::{Activity, AutoScalingGroup, LaunchTemplateSpecification},
};
use aws_sdk_ec2::types::RequestLaunchTemplateData;
use tracing::trace;

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

struct Waiter {
    start: SystemTime,
    max: Duration,

```

```
}

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

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

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

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



```

        }
    }
    Err(e) => writeln!(f, "\t\t! {e}")?,
}

Ok(())
}
}

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

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

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

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

impl ScenarioError {
    pub fn with(message: impl Into<String>) -> Self {

```



```

        ScenarioError {
            message: message.into(),
            context: None,
        }
    }

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

impl Error for ScenarioError {
    // While `Error` can capture `source` information about the underlying error,
    // for this example
    // the ScenarioError captures the underlying information in MetadataError and
    // treats it as a
    // single Error from this Crate. In other contexts, it may be appropriate to
    // model the error
    // as including the SdkError as its source.
}

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

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

        let auto_scaling_group_name = String::from(AUTOSCALING_GROUP_NAME);

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

```

```

    }

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

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

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

```

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

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

    if let Err(err) = ec2
        .delete_launch_template()
        .launch_template_id(launch_template_arn.clone())
        .send()
        .await
    {
```

```

        errs.push(ScenarioError::new(
            "Failed to clean up launch template",
            &err,
        ));
    }
    return Err(errs);
}

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

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

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

}

pub async fn clean_scenario(self) -> Result<(), Vec<ScenarioError>> {
    let _ = self.wait_for_no_scaling().await;
}

```

```

let delete_group = self
    .autoscaling
    .delete_auto_scaling_group()
    .auto_scaling_group_name(self.auto_scaling_group_name.clone())
    .send()
    .await;

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

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

if early_exit.is_err() {
    early_exit
} else {
    // Wait for delete_group to finish
    let waiter = Waiter::new();
    let mut errors = Vec::<ScenarioError>::new();
    while errors.len() < 3 {
        if let Err(e) = waiter.sleep().await {
            errors.push(e);
            continue;
        }
    }
}

```

```

        let describe_group = self
            .autoscaling
            .describe_auto_scaling_groups()
            .auto_scaling_group_names(self.auto_scaling_group_name.clone())
            .send()
            .await;
        match describe_group {
            Ok(group) => match group.auto_scaling_groups().first() {
                Some(group) => {
                    if group.status() != Some("Delete in progress") {
                        errors.push(ScenarioError::with(format!(
                            "Group in an unknown state while deleting: {}",
                            group.status().unwrap_or("unknown error")
                        )));
                        return Err(errors);
                    }
                }
                None => return Ok(()),
            },
            Err(err) => {
                errors.push(ScenarioError::new("Failed to describe
autoscaling group during cleanup 3 times, last error", &err));
            }
        }
        if errors.len() > 3 {
            return Err(errors);
        }
    }
    Err(vec![ScenarioError::with(
        "Exited cleanup wait loop without 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::

```

```

    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)
}
```

- Per informazioni dettagliate sulle API, consulta i seguenti argomenti nella Documentazione di riferimento delle API SDK AWS per Rust.
 - [CreateAutoScalingGroup](#)
 - [DeleteAutoScalingGroup](#)
 - [DescribeAutoScalingGroups](#)
 - [DescribeAutoScalingInstances](#)
 - [DescribeScalingActivities](#)
 - [DisableMetricsCollection](#)
 - [EnableMetricsCollection](#)
 - [SetDesiredCapacity](#)
 - [TerminateInstanceInAutoScalingGroup](#)
 - [UpdateAutoScalingGroup](#)

Operazioni

CreateAutoScalingGroup

Il seguente esempio di codice mostra come usare `CreateAutoScalingGroup`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta il riferimento [CreateAutoScalingGroup](#) all'API AWS SDK for Rust.

DeleteAutoScalingGroup

Il seguente esempio di codice mostra come utilizzare `DeleteAutoScalingGroup`.

SDK per Rust

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta il riferimento [DeleteAutoScalingGroup](#) all'API AWS SDK for Rust.

DescribeAutoScalingGroups

Il seguente esempio di codice mostra come utilizzare `DescribeAutoScalingGroups`.

SDK per Rust

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta il riferimento [DescribeAutoScalingGroups](#) all'API AWS SDK for Rust.

DescribeAutoScalingInstances

Il seguente esempio di codice mostra come utilizzare `DescribeAutoScalingInstances`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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))
    }


```

- Per i dettagli sulle API, consulta il riferimento [DescribeAutoScalingInstances](#) all'API AWS SDK for Rust.

DescribeScalingActivities

Il seguente esempio di codice mostra come utilizzare `DescribeScalingActivities`.

SDK per Rust

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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::

```

- Per i dettagli sulle API, consulta il riferimento [DescribeScalingActivities](#) all'API AWS SDK for Rust.

DisableMetricsCollection

Il seguente esempio di codice mostra come utilizzare `DisableMetricsCollection`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
// 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;
```

- Per i dettagli sulle API, consulta il riferimento [DisableMetricsCollection](#) all'API AWS SDK for Rust.

EnableMetricsCollection

Il seguente esempio di codice mostra come utilizzare `EnableMetricsCollection`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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;
```

- Per i dettagli sulle API, consulta il riferimento [EnableMetricsCollection](#) all'API AWS SDK for Rust.

SetDesiredCapacity

Il seguente esempio di codice mostra come utilizzare `SetDesiredCapacity`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta il riferimento [SetDesiredCapacity](#) all'API AWS SDK for Rust.

TerminateInstanceInAutoScalingGroup

Il seguente esempio di codice mostra come utilizzare `TerminateInstanceInAutoScalingGroup`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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())
}

```

- Per i dettagli sulle API, consulta il riferimento [TerminateInstanceInAutoScalingGroup](#) all'API AWS SDK for Rust.

UpdateAutoScalingGroup

Il seguente esempio di codice mostra come utilizzare `UpdateAutoScalingGroup`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta il riferimento [UpdateAutoScalingGroup](#) all'API AWS SDK for Rust.

Esempi di Amazon Bedrock Runtime con SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l'AWS SDK per Rust con Amazon Bedrock Runtime.

Ogni esempio include un collegamento al codice sorgente completo, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Anthropic Claude](#)

Anthropic Claude

conversare

Il seguente esempio di codice mostra come inviare un messaggio di testo a Anthropic Claude, utilizzando l'API Converse di Bedrock.

SDK per Rust

Note

C'è di più su. [GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.](#)

Invia un messaggio di testo a Anthropic Claude, utilizzando l'API Converse di Bedrock.

```
#[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)
}

```

Usa istruzioni, utilità Error e costanti.

```

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",
        })
    }
}
}

```

- Per i dettagli sulle API, consulta [Converse](#) in AWS SDK for Rust API reference.

ConverseStream

Il seguente esempio di codice mostra come inviare un messaggio di testo a Anthropic Claude, utilizzando l'API Converse di Bedrock ed elaborare il flusso di risposta in tempo reale.

SDK per Rust

Note

C'è di più su. [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Invia un messaggio di testo a Anthropic Claude e trasmetti in streaming i token di risposta, utilizzando l'API di Bedrock. `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(),
    })
}

```

Usa istruzioni, utilità `Error` e costanti.

```

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

- Per i dettagli sull'API, consulta la [ConverseStream](#) guida di riferimento all'API AWS SDK for Rust.

Scenario: utilizzo dello strumento con l'API Converse

Il seguente esempio di codice mostra come creare un'interazione tipica tra un'applicazione, un modello di intelligenza artificiale generativa e strumenti connessi o come APIs mediare le interazioni tra l'IA e il mondo esterno. Utilizza l'esempio del collegamento di un'API meteorologica esterna al modello di intelligenza artificiale in modo che possa fornire informazioni meteorologiche in tempo reale basate sull'input dell'utente.

SDK per Rust

Note

C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Lo scenario e la logica principali della demo. Questo orchestra la conversazione tra l'utente, l'API Amazon Bedrock Converse e uno strumento meteorologico.

```
#[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();
}

```

Lo strumento meteorologico utilizzato dalla demo. Questo script definisce le specifiche dello strumento e implementa la logica per recuperare i dati meteorologici utilizzando l'API Open-Meteo.

```
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 = request::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()?)
}

```

Utilità per stampare i blocchi di contenuto dei messaggi.

```

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:?})"
        )))
    }
}

```

Utilizza istruzioni, utilità Error e costanti.

```

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(),
        })
    }
}
```

- Per i dettagli sulle API, consulta [Converse](#) in AWS SDK for Rust API reference.

Esempi di Amazon Cognito Identity Provider che utilizzano SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l' AWS SDK per Rust con Amazon Cognito Identity Provider.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Operazioni](#)

Operazioni

ListUserPools

Il seguente esempio di codice mostra come utilizzare `ListUserPools`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta il riferimento [ListUserPools](#) all'API AWS SDK for Rust.

Esempi di Amazon Cognito Sync con SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l' AWS SDK per Rust con Amazon Cognito Sync.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Operazioni](#)

Operazioni

ListIdentityPoolUsage

Il seguente esempio di codice mostra come utilizzare `ListIdentityPoolUsage`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta il riferimento [ListIdentityPoolUsage](#) all'API AWS SDK for Rust.

Esempi di Firehose che utilizzano SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l' AWS SDK per Rust con Firehose.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Operazioni](#)

Operazioni

PutRecordBatch

Il seguente esempio di codice mostra come utilizzare `PutRecordBatch`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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
}
```

- Per i dettagli sulle API, consulta il riferimento [PutRecordBatch](#) all'API AWS SDK for Rust.

Esempi di Amazon DocumentDB con SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l'AWS SDK per Rust con Amazon DocumentDB.

Ogni esempio include un collegamento al codice sorgente completo, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Esempi serverless](#)

Esempi serverless

Richiamare una funzione Lambda da un trigger Amazon DocumentDB

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un flusso di modifiche di DocumentDB. La funzione recupera il payload DocumentDB e registra il contenuto del record.

SDK per Rust

Note

C'è di più su. [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Amazon DocumentDB con Lambda tramite Ruby.

```
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(())
}
```

Esempi di DynamoDB con SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l' AWS SDK per Rust con DynamoDB.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Gli scenari sono esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

AWS i contributi della community sono esempi che sono stati creati e gestiti da diversi team. AWS Per fornire feedback, utilizza il meccanismo fornito negli archivi collegati.

Ogni esempio include un collegamento al codice sorgente completo, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Operazioni](#)
- [Scenari](#)
- [Esempi serverless](#)
- [AWS contributi della comunità](#)

Operazioni

CreateTable

Il seguente esempio di codice mostra come utilizzare `CreateTable`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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))
    }
}
}

```

- Per i dettagli sulle API, consulta il riferimento [CreateTable](#) all'API AWS SDK for Rust.

DeleteItem

Il seguente esempio di codice mostra come utilizzare `DeleteItem`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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)),
    }
}
```

- Per i dettagli sulle API, consulta il riferimento [DeleteItem](#) all'API AWS SDK for Rust.

DeleteTable

Il seguente esempio di codice mostra come utilizzare `DeleteTable`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sulle API, consulta il riferimento [DeleteTable](#) all'API AWS SDK for Rust.

ListTables

Il seguente esempio di codice mostra come utilizzare `ListTables`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

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

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

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

Determina se esiste una tabella.

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

- Per i dettagli sulle API, consulta il riferimento [ListTables](#) all'API AWS SDK for Rust.

PutItem

Il seguente esempio di codice mostra come utilizzare `PutItem`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
pub async fn add_item(client: &Client, item: Item, table: &String) ->
    Result<ItemOut, Error> {
```

```

let user_av = AttributeValue::S(item.username);
let type_av = AttributeValue::S(item.p_type);
let age_av = AttributeValue::S(item.age);
let first_av = AttributeValue::S(item.first);
let last_av = AttributeValue::S(item.last);

let request = client
    .put_item()
    .table_name(table)
    .item("username", user_av)
    .item("account_type", type_av)
    .item("age", age_av)
    .item("first_name", first_av)
    .item("last_name", last_av);

println!("Executing request [{request:?}] to add item...");

let resp = request.send().await?;

let attributes = resp.attributes().unwrap();

let username = attributes.get("username").cloned();
let first_name = attributes.get("first_name").cloned();
let last_name = attributes.get("last_name").cloned();
let age = attributes.get("age").cloned();
let p_type = attributes.get("p_type").cloned();

println!(
    "Added user {:?}, {:?} {:?}, age {:?} as {:?} user",
    username, first_name, last_name, age, p_type
);

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

```

- Per i dettagli sulle API, consulta il riferimento [PutItem](#) all'API AWS SDK for Rust.

Query

Il seguente esempio di codice mostra come utilizzare `Query`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Trova i filmati realizzati nell'anno specificato.

```
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![])
    }
}
```

- Per ulteriori informazioni sulle API, consulta [Query](#) nella Documentazione di riferimento delle API AWS .

Scan

Il seguente esempio di codice mostra come usare `Scan`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per informazioni dettagliate sulle API, consulta [Scan](#) nella Documentazione di riferimento degli SDK AWS per l'API Rust.

Scenari

Connect a un'istanza locale

Il seguente esempio di codice mostra come sovrascrivere l'URL di un endpoint per connettersi a una distribuzione di sviluppo locale di DynamoDB e un SDK. AWS

Per ulteriori informazioni, consulta [DynamoDB Local](#).

SDK per Rust

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

Creazione di un'applicazione serverless per gestire foto

Nell'esempio di codice seguente viene illustrato come creare un'applicazione serverless che consente agli utenti di gestire le foto mediante etichette.

SDK per Rust

Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su [GitHub](#).

Per approfondire l'origine di questo esempio, consulta il post su [AWS Community](#).

Servizi utilizzati in questo esempio


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

Esecuzione di una query mediante PartiQL

L'esempio di codice seguente mostra come:

- Ricezione di un articolo eseguendo un'istruzione SELECT.
- Aggiunta di un elemento eseguendo un'istruzione INSERT.
- Aggiornamento di un elemento eseguendo un'istruzione UPDATE.
- Eliminazione di un elemento eseguendo un'istruzione DELETE.

SDK per Rust

 Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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 "{table}" WHERE "{key}" = ?"#))
        .set_parameters(Some(vec![AttributeValue::S(value)]))
        .send()
        .await?;

    println!("Deleted item.");

    Ok(())
}

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

    Ok(())
}
}

```

- Per i dettagli sulle API, consulta il riferimento [ExecuteStatement](#) all'API AWS SDK for Rust.

Salvataggio di EXIF e altre informazioni sull'immagine

L'esempio di codice seguente mostra come:

- Recuperare informazioni EXIF da un file JPG, JPEG o PNG.
- Carica il file immagine in un bucket Amazon S3.
- Utilizza Amazon Rekognition per identificare i tre attributi principali (etichette) nel file.
- Aggiungi le informazioni su EXIF ed etichette a una tabella Amazon DynamoDB nella regione.

SDK per Rust

Recupera le informazioni EXIF da un file JPG, JPEG o PNG, carica il file di immagine in un bucket Amazon S3, utilizza Amazon Rekognition per identificare i tre attributi principali (etichette in Amazon Rekognition) nel file e aggiungi le informazioni su EXIF ed etichette a una tabella Amazon DynamoDB nella regione.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su [GitHub](#).

Servizi utilizzati in questo esempio

- DynamoDB
- Amazon Rekognition
- Amazon S3

Esempi serverless

Richiamare una funzione Lambda da un trigger DynamoDB

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un flusso DynamoDB. La funzione recupera il payload DocumentDB e registra il contenuto del record.

SDK per Rust

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento DynamoDB con Lambda tramite 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(())
}
```

Segnalazione di errori di elementi batch per funzioni Lambda con un trigger DynamoDB

Il seguente esempio di codice mostra come implementare una risposta batch parziale per le funzioni Lambda che ricevono eventi da un flusso DynamoDB. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

SDK per Rust

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite 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 contributi della comunità

Crea e testa un'applicazione serverless

Il seguente esempio di codice mostra come creare e testare un'applicazione serverless utilizzando API Gateway con Lambda e DynamoDB.

SDK per Rust

Mostra come creare e testare un'applicazione serverless composta da un API Gateway con Lambda e DynamoDB utilizzando Rust SDK.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. [GitHub](#)

Servizi utilizzati in questo esempio

- API Gateway
- DynamoDB
- Lambda

Esempi di Amazon EBS con SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l' AWS SDK per Rust con Amazon EBS.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Operazioni](#)

Operazioni

CompleteSnapshot

Il seguente esempio di codice mostra come utilizzare `CompleteSnapshot`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta il riferimento [CompleteSnapshot](#) all'API AWS SDK for Rust.

PutSnapshotBlock

Il seguente esempio di codice mostra come utilizzare `PutSnapshotBlock`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta il riferimento [PutSnapshotBlock](#) all'API AWS SDK for Rust.

StartSnapshot

Il seguente esempio di codice mostra come utilizzare `StartSnapshot`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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())
}
```

- Per i dettagli sulle API, consulta il riferimento [StartSnapshot](#) all'API AWS SDK for Rust.

EC2 Esempi di Amazon che utilizzano SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l'AWS SDK per Rust con Amazon EC2.

Le nozioni di base sono esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Ciao Amazon EC2

I seguenti esempi di codice mostrano come iniziare a usare Amazon EC2.

SDK per Rust

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sulle API, consulta il riferimento [DescribeSecurityGroups](#) all'API AWS SDK for Rust.

Argomenti

- [Nozioni di base](#)
- [Operazioni](#)

Nozioni di base

Informazioni di base

L'esempio di codice seguente mostra come:

- Creare una coppia di chiavi e un gruppo di sicurezza.
- Selezionare un'Amazon Machine Image (AMI) e un tipo di istanza compatibile e quindi creare un'istanza.
- Arrestare e riavviare l'istanza.
- Associazione di un indirizzo IP elastico all'istanza
- Connettiti alla tua istanza con SSH, quindi elimina le risorse.

SDK per Rust

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

L' `EC2InstanceScenario` implementazione contiene la logica per eseguire l'esempio nel suo insieme.

```

    //! 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!
    ("-----");
}

```

La struttura EC2 Impl funge da punto di automazione per i test e le sue funzioni racchiudono le chiamate EC2 SDK.

```

use std::{net::Ipv4Addr, time::Duration};

use aws_sdk_ec2::{
    client::Waiters,
    error::ProvideErrorMetadata,
    operation::{
        allocate_address::AllocateAddressOutput,
        associate_address::AssociateAddressOutput,
    },
    types::{
        DomainType, Filter, Image, Instance, InstanceType, IpPermission, IpRange,
        KeyPairInfo,
        SecurityGroup, Tag,
    },
    Client as EC2Client,
};
use aws_sdk_ssm::types::Parameter;
use aws_smithy_runtime_api::client::waiters::error::WaiterError;

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

#[cfg(not(test))]
pub use EC2Impl as EC2;

#[cfg(test)]
pub use MockEC2Impl as EC2;

#[derive(Clone)]
pub struct EC2Impl {
    pub client: EC2Client,
}

#[cfg_attr(test, automock)]
impl EC2Impl {
    pub fn new(client: EC2Client) -> Self {
        EC2Impl { client }
    }

    pub async fn create_key_pair(&self, name: String) -> Result<(KeyPairInfo,
String), EC2Error> {
        tracing::info!("Creating key pair {name}");
    }
}

```

```

    let output = self.client.create_key_pair().key_name(name).send().await?;
    let info = KeyPairInfo::builder()
        .set_key_name(output.key_name)
        .set_key_fingerprint(output.key_fingerprint)
        .set_key_pair_id(output.key_pair_id)
        .build();
    let material = output
        .key_material
        .ok_or_else(|| EC2Error::new("Create Key Pair has no key material"))?;
    Ok((info, material))
}

pub async fn list_key_pair(&self) -> Result<Vec<KeyPairInfo>, EC2Error> {
    let output = self.client.describe_key_pairs().send().await?;
    Ok(output.key_pairs.unwrap_or_default())
}

pub async fn delete_key_pair(&self, key_name: &str) -> Result<(), EC2Error> {
    let key_name: String = key_name.into();
    tracing::info!("Deleting key pair {key_name}");
    self.client
        .delete_key_pair()
        .key_name(key_name)
        .send()
        .await?;
    Ok(())
}

pub async fn create_security_group(
    &self,
    name: &str,
    description: &str,
) -> Result<SecurityGroup, EC2Error> {
    tracing::info!("Creating security group {name}");
    let create_output = self
        .client
        .create_security_group()
        .group_name(name)
        .description(description)
        .send()
        .await
        .map_err(EC2Error::from)?;

    let group_id = create_output

```



```

        .group_id
        .ok_or_else(|| EC2Error::new("Missing security group id after
creation"))?);

    let group = self
        .describe_security_group(&group_id)
        .await?
        .ok_or_else(|| {
            EC2Error::new(format!("Could not find security group with id
{group_id}"))
        })?);

    tracing::info!("Created security group {name} as {group_id}");

    Ok(group)
}

/// Find a single security group, by ID. Returns Err if multiple groups are
found.
pub async fn describe_security_group(
    &self,
    group_id: &str,
) -> Result<Option<SecurityGroup>, EC2Error> {
    let group_id: String = group_id.into();
    let describe_output = self
        .client
        .describe_security_groups()
        .group_ids(&group_id)
        .send()
        .await?;

    let mut groups = describe_output.security_groups.unwrap_or_default();

    match groups.len() {
        0 => Ok(None),
        1 => Ok(Some(groups.remove(0))),
        _ => Err(EC2Error::new(format!(
            "Expected single group for {group_id}"
        ))),
    }
}

/// Add an ingress rule to a security group explicitly allowing IPv4 address
/// as {ip}/32 over TCP port 22.

```

```

pub async fn authorize_security_group_ssh_ingress(
    &self,
    group_id: &str,
    ingress_ips: Vec<Ipv4Addr>,
) -> Result<(), EC2Error> {
    tracing::info!("Authorizing ingress for security group {group_id}");
    self.client
        .authorize_security_group_ingress()
        .group_id(group_id)
        .set_ip_permissions(Some(
            ingress_ips
                .into_iter()
                .map(|ip| {
                    IpPermission::builder()
                        .ip_protocol("tcp")
                        .from_port(22)
                        .to_port(22)
                        .ip_ranges(IpRange::builder().cidr_ip(format!(
                            "{ip}/32"))).build())
                    .build()
                })
                .collect(),
        ))
        .send()
        .await?;
    Ok(())
}

pub async fn delete_security_group(&self, group_id: &str) -> Result<(),
EC2Error> {
    tracing::info!("Deleting security group {group_id}");
    self.client
        .delete_security_group()
        .group_id(group_id)
        .send()
        .await?;
    Ok(())
}

pub async fn list_images(&self, ids: Vec<Parameter>) -> Result<Vec<Image>,
EC2Error> {
    let image_ids = ids.into_iter().filter_map(|p| p.value).collect();
    let output = self
        .client

```

```

        .describe_images()
        .set_image_ids(Some(image_ids))
        .send()
        .await?;

let images = output.images.unwrap_or_default();
if images.is_empty() {
    Err(EC2Error::new("No images for selected AMIs"))
} else {
    Ok(images)
}
}

/// List instance types that match an image's architecture and are free tier
eligible.
pub async fn list_instance_types(&self, image: &Image) ->
Result<Vec<InstanceType>, EC2Error> {
    let architecture = format!(
        "{}",
        image.architecture().ok_or_else(|| EC2Error::new(format!(
            "Image {:?} does not have a listed architecture",
            image.image_id()
        )))?
    );
    let free_tier_eligible_filter = Filter::builder()
        .name("free-tier-eligible")
        .values("false")
        .build();
    let supported_architecture_filter = Filter::builder()
        .name("processor-info.supported-architecture")
        .values(architecture)
        .build();
    let response = self
        .client
        .describe_instance_types()
        .filters(free_tier_eligible_filter)
        .filters(supported_architecture_filter)
        .send()
        .await?;

    Ok(response
        .instance_types
        .unwrap_or_default()
        .into_iter())
}

```

```

        .filter_map(|iti| iti.instance_type)
        .collect()
    }

pub async fn create_instance<'a>(
    &self,
    image_id: &'a str,
    instance_type: InstanceType,
    key_pair: &'a KeyPairInfo,
    security_groups: Vec<&'a SecurityGroup>,
) -> Result<String, EC2Error> {
    let run_instances = self
        .client
        .run_instances()
        .image_id(image_id)
        .instance_type(instance_type)
        .key_name(
            key_pair
                .key_name()
                .ok_or_else(|| EC2Error::new("Missing key name when launching
instance"))?,
        )
        .set_security_group_ids(Some(
            security_groups
                .iter()
                .filter_map(|sg| sg.group_id.clone())
                .collect(),
        ))
        .min_count(1)
        .max_count(1)
        .send()
        .await?;

    if run_instances.instances().is_empty() {
        return Err(EC2Error::new("Failed to create instance"));
    }

    let instance_id = run_instances.instances()[0].instance_id().unwrap();
    let response = self
        .client
        .create_tags()
        .resources(instance_id)
        .tags(
            Tag::builder()

```

```

        .key("Name")
        .value("From SDK Examples")
        .build(),
    )
    .send()
    .await;

match response {
    Ok(_) => tracing::info!("Created {instance_id} and applied tags."),
    Err(err) => {
        tracing::info!("Error applying tags to {instance_id}: {err:?}");
        return Err(err.into());
    }
}

tracing::info!("Instance is created.");

Ok(instance_id.to_string())
}

/// Wait for an instance to be ready and status ok (default wait 60 seconds)
pub async fn wait_for_instance_ready(
    &self,
    instance_id: &str,
    duration: Option<Duration>,
) -> Result<(), EC2Error> {
    self.client
        .wait_until_instance_status_ok()
        .instance_ids(instance_id)
        .wait(duration.unwrap_or(Duration::from_secs(60)))
        .await
        .map_err(|err| match err {
            WaiterError::ExceededMaxWait(exceeded) => EC2Error(format!(
                "Exceeded max time ({}s) waiting for instance to start.",
                exceeded.max_wait().as_secs()
            )),
            _ => EC2Error::from(err),
        })?;
    Ok(())
}

pub async fn describe_instance(&self, instance_id: &str) -> Result<Instance,
EC2Error> {
    let response = self

```

```

        .client
        .describe_instances()
        .instance_ids(instance_id)
        .send()
        .await?;

    let instance = response
        .reservations()
        .first()
        .ok_or_else(|| EC2Error::new(format!("No instance reservations for
{instance_id}")))?
        .instances()
        .first()
        .ok_or_else(|| {
            EC2Error::new(format!("No instances in reservation for
{instance_id}"))
        })?;

    Ok(instance.clone())
}

pub async fn start_instance(&self, instance_id: &str) -> Result<(), EC2Error> {
    tracing::info!("Starting instance {instance_id}");

    self.client
        .start_instances()
        .instance_ids(instance_id)
        .send()
        .await?;

    tracing::info!("Started instance.");

    Ok(())
}

pub async fn stop_instance(&self, instance_id: &str) -> Result<(), EC2Error> {
    tracing::info!("Stopping instance {instance_id}");

    self.client
        .stop_instances()
        .instance_ids(instance_id)
        .send()
        .await?;
}

```

```

        self.wait_for_instance_stopped(instance_id, None).await?;

        tracing::info!("Stopped instance.");

        Ok(())
    }

    pub async fn reboot_instance(&self, instance_id: &str) -> Result<(), EC2Error> {
        tracing::info!("Rebooting instance {instance_id}");

        self.client
            .reboot_instances()
            .instance_ids(instance_id)
            .send()
            .await?;

        Ok(())
    }

    pub async fn wait_for_instance_stopped(
        &self,
        instance_id: &str,
        duration: Option<Duration>,
    ) -> Result<(), EC2Error> {
        self.client
            .wait_until_instance_stopped()
            .instance_ids(instance_id)
            .wait(duration.unwrap_or(Duration::from_secs(60)))
            .await
            .map_err(|err| match err {
                WaiterError::ExceededMaxWait(exceeded) => EC2Error(format!(
                    "Exceeded max time ({}s) waiting for instance to stop.",
                    exceeded.max_wait().as_secs(),
                )),
                _ => EC2Error::from(err),
            })?;

        Ok(())
    }

    pub async fn delete_instance(&self, instance_id: &str) -> Result<(), EC2Error> {
        tracing::info!("Deleting instance with id {instance_id}");
        self.stop_instance(instance_id).await?;
        self.client
            .terminate_instances()

```

```

        .instance_ids(instance_id)
        .send()
        .await?;
    self.wait_for_instance_terminated(instance_id).await?;
    tracing::info!("Terminated instance with id {instance_id}");
    Ok(())
}

    async fn wait_for_instance_terminated(&self, instance_id: &str) -> Result<(),
    EC2Error> {
        self.client
            .wait_until_instance_terminated()
            .instance_ids(instance_id)
            .wait(Duration::from_secs(60))
            .await
            .map_err(|err| match err {
                WaiterError::ExceededMaxWait(exceeded) => EC2Error(format!(
                    "Exceeded max time ({}s) waiting for instance to terminate.",
                    exceeded.max_wait().as_secs(),
                )),
                _ => EC2Error::from(err),
            })?;
        Ok(())
    }

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

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

```



```

pub async fn associate_ip_address(
    &self,
    allocation_id: &str,
    instance_id: &str,
) -> Result<AssociateAddressOutput, EC2Error> {
    let response = self
        .client
        .associate_address()
        .allocation_id(allocation_id)
        .instance_id(instance_id)
        .send()
        .await?;
    Ok(response)
}

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

#[derive(Debug)]
pub struct EC2Error(String);
impl EC2Error {
    pub fn new(value: impl Into<String>) -> Self {
        EC2Error(value.into())
    }

    pub fn add_message(self, message: impl Into<String>) -> Self {
        EC2Error(format!("{}: {}", message.into(), self.0))
    }
}

impl<T: ProvideErrorMetadata> From<T> for EC2Error {
    fn from(value: T) -> Self {
        EC2Error(format!(
            "{}: {}",
            value

```

```

        .code()
        .map(String::from)
        .unwrap_or("unknown code".into()),
    value
        .message()
        .map(String::from)
        .unwrap_or("missing reason".into()),
    ))
}
}

impl std::error::Error for EC2Error {}

impl std::fmt::Display for EC2Error {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(f, "{}", self.0)
    }
}
}

```

La struttura SSM funge da punto di automazione per i test e le sue funzioni racchiudono le chiamate SSM SDK.

```

use aws_sdk_ssm::{types::Parameter, Client};
use aws_smithy_async::future::pagination_stream::TryFlatMap;

use crate::ec2::EC2Error;

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

#[cfg(not(test))]
pub use SSMImpl as SSM;

#[cfg(test)]
pub use MockSSMImpl as SSM;

pub struct SSMImpl {
    inner: Client,
}

#[cfg_attr(test, automock)]

```

```

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

    pub async fn list_path(&self, path: &str) -> Result<Vec<Parameter>, EC2Error> {
        let maybe_params: Vec<Result<Parameter, _>> = TryFlatMap::new(
            self.inner
                .get_parameters_by_path()
                .path(path)
                .into_paginator()
                .send(),
        )
        .flat_map(|item| item.parameters.unwrap_or_default())
        .collect()
        .await;
        // Fail on the first error
        let params = maybe_params
            .into_iter()
            .collect:::<Result<Vec<Parameter>, _>>()?;
        Ok(params)
    }
}

```

Lo scenario utilizza diverse strutture in stile «Manager» per gestire l'accesso alle risorse che vengono create ed eliminate durante lo scenario.

```

use aws_sdk_ec2::operation::{
    allocate_address::AllocateAddressOutput,
    associate_address::AssociateAddressOutput,
};

use crate::ec2::{EC2Error, EC2};

/// ElasticIpManager tracks the lifecycle of a public IP address, including its
/// allocation from the global pool and association with a specific instance.
#[derive(Debug, Default)]
pub struct ElasticIpManager {
    elastic_ip: Option<AllocateAddressOutput>,
    association: Option<AssociateAddressOutput>,
}

```

```

impl ElasticIpManager {
    pub fn has_allocation(&self) -> bool {
        self.elastic_ip.is_some()
    }

    pub fn public_ip(&self) -> &str {
        if let Some(allocation) = &self.elastic_ip {
            if let Some(addr) = allocation.public_ip() {
                return addr;
            }
        }
        "0.0.0.0"
    }

    pub async fn allocate(&mut self, ec2: &EC2) -> Result<(), EC2Error> {
        let allocation = ec2.allocate_ip_address().await?;
        self.elastic_ip = Some(allocation);
        Ok(())
    }

    pub async fn associate(&mut self, ec2: &EC2, instance_id: &str) -> Result<(),
    EC2Error> {
        if let Some(allocation) = &self.elastic_ip {
            if let Some(allocation_id) = allocation.allocation_id() {
                let association = ec2.associate_ip_address(allocation_id,
instance_id).await?;
                self.association = Some(association);
                return Ok(());
            }
        }
        Err(EC2Error::new("No ip address allocation to associate"))
    }

    pub async fn remove(mut self, ec2: &EC2) -> Result<(), EC2Error> {
        if let Some(association) = &self.association {
            if let Some(association_id) = association.association_id() {
                ec2.disassociate_ip_address(association_id).await?;
            }
        }
        self.association = None;
        if let Some(allocation) = &self.elastic_ip {
            if let Some(allocation_id) = allocation.allocation_id() {
                ec2.deallocate_ip_address(allocation_id).await?;
            }
        }
    }
}

```

```
    }
    }
    self.elastic_ip = None;
    Ok(())
}
}

use std::fmt::Display;

use aws_sdk_ec2::types::{Instance, InstanceType, KeyPairInfo, SecurityGroup};

use crate::ec2::{EC2Error, EC2};

/// InstanceManager wraps the lifecycle of an EC2 Instance.
#[derive(Debug, Default)]
pub struct InstanceManager {
    instance: Option<Instance>,
}

impl InstanceManager {
    pub fn instance_id(&self) -> &str {
        if let Some(instance) = &self.instance {
            if let Some(id) = instance.instance_id() {
                return id;
            }
        }
        "Unknown"
    }

    pub fn instance_name(&self) -> &str {
        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."),
        }
    }
}
}

```

Il punto di ingresso principale per lo scenario.

```
use ec2_code_examples::{
    ec2::EC2,
    getting_started::{
        scenario::{run, Ec2InstanceScenario},
        util::UtilImpl,
    },
    ssm::SSM,
};

#[tokio::main]
async fn main() {
    tracing_subscriber::fmt::init();
    let sdk_config = aws_config::load_from_env().await;
    let ec2 = EC2::new(aws_sdk_ec2::Client::new(&sdk_config));
    let ssm = SSM::new(aws_sdk_ssm::Client::new(&sdk_config));
    let util = UtilImpl {};
    let scenario = Ec2InstanceScenario::new(ec2, ssm, util);
    run(scenario).await;
}
```

- Per informazioni dettagliate sulle API, consulta i seguenti argomenti nella Documentazione di riferimento delle API SDK AWS per Rust.
 - [AllocateAddress](#)
 - [AssociateAddress](#)
 - [AuthorizeSecurityGroupIngress](#)
 - [CreateKeyPair](#)
 - [CreateSecurityGroup](#)
 - [DeleteKeyPair](#)
 - [DeleteSecurityGroup](#)
 - [DescribeImages](#)
 - [DescribeInstanceTypes](#)
 - [DescribeInstances](#)
 - [DescribeKeyPairs](#)
 - [DescribeSecurityGroups](#)
 - [DisassociateAddress](#)
 - [ReleaseAddress](#)

- [RunInstances](#)
- [StartInstances](#)
- [StopInstances](#)
- [TerminateInstances](#)
- [UnmonitorInstances](#)

Operazioni

AllocateAddress

Il seguente esempio di codice mostra come utilizzare `AllocateAddress`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).


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

- Per i dettagli sulle API, consulta il riferimento [AllocateAddress](#) all'API AWS SDK for Rust.

AssociateAddress

Il seguente esempio di codice mostra come utilizzare `AssociateAddress`.

SDK per Rust

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).


```
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)
}
```

- Per i dettagli sulle API, consulta il riferimento [AssociateAddress](#) all'API AWS SDK for Rust.

AuthorizeSecurityGroupIngress

Il seguente esempio di codice mostra come utilizzare `AuthorizeSecurityGroupIngress`.

SDK per Rust

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// 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(())
}


```

- Per i dettagli sulle API, consulta il riferimento [AuthorizeSecurityGroupIngress](#) all'API AWS SDK for Rust.

CreateKeyPair

Il seguente esempio di codice mostra come utilizzare `CreateKeyPair`.

SDK per Rust

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Implementazione di Rust che chiama la `create_key_pair` del EC2 Client ed estrae il materiale restituito.

```
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))
}
```

Una funzione che chiama l'impl `create_key` e salva in modo sicuro la chiave privata PEM.

```
/// 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)
    }

```

- Per i dettagli sulle API, consulta la guida di riferimento all'API SDK for [CreateKeyPairRust](#) AWS.

CreateSecurityGroup

Il seguente esempio di codice mostra come utilizzare `CreateSecurityGroup`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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)
}
```

- Per i dettagli sulle API, consulta il riferimento [CreateSecurityGroup](#) all'API AWS SDK for Rust.

CreateTags

Il seguente esempio di codice mostra come utilizzare `CreateTags`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Questo esempio applica il tag Name dopo aver creato un'istanza.

```
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")
        )
    .send()
    .await?;
}
```

```

        .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())
}

```

- Per i dettagli sull'API, consulta il riferimento [CreateTags](#) all'API AWS SDK for Rust.

DeleteKeyPair

Il seguente esempio di codice mostra come utilizzare `DeleteKeyPair`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Wrapper around `delete_key` che rimuove anche la chiave PEM privata di supporto.

```

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(())  
}
```

- Per i dettagli sulle API, consulta il riferimento all'API SDK for [DeleteKeyPair](#) Rust AWS .

DeleteSecurityGroup

Il seguente esempio di codice mostra come utilizzare `DeleteSecurityGroup`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())  
}
```

```
}
```

- Per i dettagli sulle API, consulta il riferimento [DeleteSecurityGroup](#) all'API AWS SDK for Rust.

DeleteSnapshot

Il seguente esempio di codice mostra come utilizzare `DeleteSnapshot`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

    println!("Deleted");

    Ok(())
}
```

- Per i dettagli sulle API, consulta il riferimento [DeleteSnapshot](#) all'API AWS SDK for Rust.

DescribeImages

Il seguente esempio di codice mostra come utilizzare `DescribeImages`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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)
    }
}

```

Utilizzo della funzione `list_images` con SSM per limitare in base all'ambiente. Per maggiori dettagli su SSM, consulta `_ssm_` section.html. <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)
}

```

- Per i dettagli sulle API, consulta la guida di riferimento [DescribeImages](#) all'API AWS SDK for Rust.

DescribeInstanceStatus

Il seguente esempio di codice mostra come utilizzare `DescribeInstanceStatus`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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(())
}

```

- Per i dettagli sulle API, consulta il riferimento [DescribeInstanceStatus](#) all'API AWS SDK for Rust.

DescribeInstanceTypes

Il seguente esempio di codice mostra come utilizzare `DescribeInstanceTypes`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// 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())
}

```

- Per i dettagli sulle API, consulta il riferimento [DescribeInstanceTypes](#) all'API AWS SDK for Rust.

DescribeInstances

Il seguente esempio di codice mostra come utilizzare `DescribeInstances`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Recupera i dettagli di un' EC2 istanza.

```

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())
}

```

Dopo aver creato un' EC2 istanza, recuperate e memorizzate i relativi dettagli.

```

/// 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(())
}

```

- Per i dettagli sull'API, consulta la [DescribeInstances](#) guida di riferimento all'API AWS SDK for Rust.

DescribeKeyPairs

Il seguente esempio di codice mostra come utilizzare `DescribeKeyPairs`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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())
}
```

- Per i dettagli sulle API, consulta il riferimento [DescribeKeyPairs](#) all'API AWS SDK for Rust.

DescribeRegions

Il seguente esempio di codice mostra come utilizzare `DescribeRegions`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())  
}
```

- Per i dettagli sulle API, consulta il riferimento [DescribeRegions](#) all'API AWS SDK for Rust.

DescribeSecurityGroups

Il seguente esempio di codice mostra come utilizzare `DescribeSecurityGroups`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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}");
    }
}
}

```

- Per i dettagli sulle API, consulta il riferimento [DescribeSecurityGroups](#) all'API AWS SDK for Rust.

DescribeSnapshots

Il seguente esempio di codice mostra come utilizzare `DescribeSnapshots`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Mostra lo stato di uno snapshot.

```

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(())
}
```

- Per i dettagli sulle API, consulta il riferimento [DescribeSnapshots](#) all'API AWS SDK for Rust.

DisassociateAddress

Il seguente esempio di codice mostra come utilizzare `DisassociateAddress`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).


```

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

```

- Per i dettagli sulle API, consulta il riferimento [DisassociateAddress](#) all'API AWS SDK for Rust.

RebootInstances

Il seguente esempio di codice mostra come utilizzare `RebootInstances`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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(())
    }

```

I camerieri, ad esempio, devono trovarsi negli stati stop e ready, utilizzando l'API Waiters. L'utilizzo dell'API Waiters richiede `use aws_sdk_ec2: :client: :Waiters` nel file rust.`

```

/// Wait for an instance to be ready and status ok (default wait 60 seconds)
pub async fn wait_for_instance_ready(
    &self,
    instance_id: &str,
    duration: Option<Duration>,
) -> Result<(), EC2Error> {
    self.client
        .wait_until_instance_status_ok()
        .instance_ids(instance_id)
        .wait(duration.unwrap_or(Duration::from_secs(60)))
        .await
        .map_err(|err| match err {
            WaiterError::ExceededMaxWait(exceeded) => EC2Error(format!(
                "Exceeded max time ({}s) waiting for instance to start.",
                exceeded.max_wait().as_secs()
            )),
            _ => EC2Error::from(err),
        })?;
    Ok(())
}

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(())
}

```

- Per i dettagli [RebootInstances](#) sull'API AWS , consulta il riferimento all'API SDK for Rust.

ReleaseAddress

Il seguente esempio di codice mostra come utilizzare `ReleaseAddress`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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

```

- Per i dettagli sulle API, consulta il riferimento [ReleaseAddress](#) all'API AWS SDK for Rust.

RunInstances

Il seguente esempio di codice mostra come utilizzare `RunInstances`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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())
}
```

- Per i dettagli sulle API, consulta il riferimento [RunInstances](#) all'API AWS SDK for Rust.

StartInstances

Il seguente esempio di codice mostra come utilizzare `StartInstances`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Avvia un' EC2 istanza per ID di istanza.

```
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(())
}
```

Attendi che un'istanza sia negli stati ready e status ok, utilizzando l'API Waiters. L'utilizzo dell'API Waiters richiede `use aws_sdk_ec2: :client: :Waiters` nel file rust.

```
/// Wait for an instance to be ready and status ok (default wait 60 seconds)
pub async fn wait_for_instance_ready(
    &self,
    instance_id: &str,
    duration: Option<Duration>,
) -> Result<(), EC2Error> {
    self.client
        .wait_until_instance_status_ok()
        .instance_ids(instance_id)
        .wait(duration.unwrap_or(Duration::from_secs(60)))
        .await
        .map_err(|err| match err {
            WaiterError::ExceededMaxWait(exceeded) => EC2Error(format!(
                "Exceeded max time ({}s) waiting for instance to start.",
                exceeded.max_wait().as_secs()
            )),
            _ => EC2Error::from(err),
        })?;
    Ok(())
}
```

- Per i dettagli [StartInstances](#) sull'API AWS , consulta il riferimento all'API SDK for Rust.

StopInstances

Il seguente esempio di codice mostra come utilizzare `StopInstances`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

Attendi che un'istanza si trovi nello stato interrotto, utilizzando l'API Waiters. L'utilizzo dell'API Waiters richiede `use aws_sdk_ec2: :client: :Waiters` nel file rust.

```
pub async fn stop_instance(&self, instance_id: &str) -> Result<(), EC2Error> {
    tracing::info!("Stopping instance {instance_id}");

    self.client
        .stop_instances()
        .instance_ids(instance_id)
        .send()
        .await?;

    self.wait_for_instance_stopped(instance_id, None).await?;
```

```

        tracing::info!("Stopped instance.");

        Ok(())
    }

```

- Per i dettagli [StopInstances](#) sull'API AWS , consulta il riferimento all'API SDK for Rust.

TerminateInstances

Il seguente esempio di codice mostra come utilizzare `TerminateInstances`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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(())
}

```

Attendi che un'istanza sia terminata, utilizzando l'API Waiters. L'utilizzo dell'API Waiters richiede ``use aws_sdk_ec2: :client: :Waiters`` nel file rust.

```

    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(())
}
```

- Per i dettagli [TerminateInstances](#) sull'API AWS , consulta il riferimento all'API SDK for Rust.

Esempi di Amazon ECR con SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l' AWS SDK per Rust con Amazon ECR.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti


- [Operazioni](#)

Operazioni

DescribeRepositories

Il seguente esempio di codice mostra come utilizzare `DescribeRepositories`.

SDK per Rust

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta il riferimento [DescribeRepositories](#) all'API AWS SDK for Rust.

ListImages

Il seguente esempio di codice mostra come utilizzare `ListImages`.

SDK per Rust

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta il riferimento [ListImages](#) all'API AWS SDK for Rust.

Esempi di Amazon ECS con SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l' AWS SDK per Rust con Amazon ECS.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Operazioni](#)

Operazioni

CreateCluster

Il seguente esempio di codice mostra come utilizzare `CreateCluster`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta il riferimento [CreateCluster](#) all'API AWS SDK for Rust.

DeleteCluster

Il seguente esempio di codice mostra come utilizzare `DeleteCluster`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}

```

- Per i dettagli sulle API, consulta il riferimento [DeleteCluster](#) all'API AWS SDK for Rust.

DescribeClusters

Il seguente esempio di codice mostra come utilizzare `DescribeClusters`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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(())
}

```

- Per i dettagli sulle API, consulta il riferimento [DescribeClusters](#) all'API AWS SDK for Rust.

Esempi di Amazon EKS che utilizzano SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l' AWS SDK per Rust con Amazon EKS.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Operazioni](#)

Operazioni

CreateCluster

Il seguente esempio di codice mostra come utilizzare `CreateCluster`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta il riferimento [CreateCluster](#) all'API AWS SDK for Rust.

DeleteCluster

Il seguente esempio di codice mostra come utilizzare `DeleteCluster`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta il riferimento [DeleteCluster](#) all'API AWS SDK for Rust.

AWS Glue esempi che utilizzano SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l' AWS SDK per Rust con. AWS Glue

Le nozioni di base sono esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Ciao AWS Glue

L'esempio di codice seguente mostra come iniziare a utilizzare AWS Glue.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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)),
    }
}
```



```
}
```

- Per i dettagli sulle API, consulta il riferimento [ListJobs](#) all'API AWS SDK for Rust.

Argomenti

- [Nozioni di base](#)
- [Operazioni](#)

Nozioni di base

Informazioni di base

L'esempio di codice seguente mostra come:

- Crea un crawler che esegue la scansione di un bucket Amazon S3 pubblico e genera un database di metadati in formato CSV.
- Elenca le informazioni su database e tabelle nel tuo AWS Glue Data Catalog.
- Crea un processo per estrarre i dati CSV dal bucket S3, trasformare i dati e caricare l'output in formato JSON in un altro bucket S3.
- Elenca le informazioni sulle esecuzioni dei processi, visualizza i dati trasformati e pulisci le risorse.

Per ulteriori informazioni, consulta [Tutorial: Guida introduttiva a AWS Glue Studio](#).

SDK per Rust

Note

C'è altro da sapere GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Creare e avviare un crawler in grado di eseguire il crawling di un bucket pubblico di Amazon Simple Storage Service (Amazon S3) generando un database di metadati che descrive i dati rilevati in formato CSV.

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

```

Elenca le informazioni su database e tabelle nel tuo 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();

```

Creare e avviare un processo che estrae i dati CSV dal bucket Amazon S3 di origine, li trasforma rimuovendo e rinominando i campi e carica l'output in formato JSON in un altro bucket Amazon S3.

```

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();
```

Eliminare tutte le risorse create dalla demo.

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

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

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

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

- Per informazioni dettagliate sulle API, consulta i seguenti argomenti nella Documentazione di riferimento delle API SDK AWS per Rust.
 - [CreateCrawler](#)
 - [CreateJob](#)
 - [DeleteCrawler](#)
 - [DeleteDatabase](#)
 - [DeleteJob](#)
 - [DeleteTable](#)
 - [GetCrawler](#)
 - [GetDatabase](#)
 - [GetDatabases](#)
 - [GetJob](#)
 - [GetJobRun](#)
 - [GetJobRuns](#)
 - [GetTables](#)
 - [ListJobs](#)
 - [StartCrawler](#)
 - [StartJobRun](#)

Operazioni

CreateCrawler

Il seguente esempio di codice mostra come utilizzare `CreateCrawler`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sulle API, consulta il riferimento [CreateCrawler](#) all'API AWS SDK for Rust.

CreateJob

Il seguente esempio di codice mostra come utilizzare `CreateJob`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sulle API, consulta la [CreateJob](#) guida di riferimento all'API AWS SDK for Rust.

DeleteCrawler

Il seguente esempio di codice mostra come utilizzare `DeleteCrawler`.

SDK per Rust

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).


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

- Per i dettagli sulle API, consulta la [DeleteCrawler](#) guida di riferimento all'API AWS SDK for Rust.

DeleteDatabase

Il seguente esempio di codice mostra come utilizzare>DeleteDatabase.

SDK per Rust

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sulle API, consulta la [DeleteDatabase](#) guida di riferimento all'API AWS SDK for Rust.

DeleteJob

Il seguente esempio di codice mostra come utilizzare `DeleteJob`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sulle API, consulta la [DeleteJob](#) guida di riferimento all'API AWS SDK for Rust.

DeleteTable

Il seguente esempio di codice mostra come utilizzare `DeleteTable`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

```
}
```

- Per i dettagli sulle API, consulta la [DeleteTable](#) guida di riferimento all'API AWS SDK for Rust.

GetCrawler

Il seguente esempio di codice mostra come utilizzare `GetCrawler`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sulle API, consulta la [GetCrawler](#) guida di riferimento all'API AWS SDK for Rust.

GetDatabase

Il seguente esempio di codice mostra come utilizzare `GetDatabase`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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()))?;

```

- Per i dettagli sulle API, consulta la [GetDatabase](#) guida di riferimento all'API AWS SDK for Rust.

GetJobRun

Il seguente esempio di codice mostra come utilizzare `GetJobRun`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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();
}
```

- Per i dettagli sulle API, consulta la [GetJobRun](#) guida di riferimento all'API AWS SDK for Rust.

GetTables

Il seguente esempio di codice mostra come utilizzare `GetTables`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

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

- Per i dettagli sulle API, consulta la [GetTables](#) guida di riferimento all'API AWS SDK for Rust.

ListJobs

Il seguente esempio di codice mostra come utilizzare `ListJobs`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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)),
    }
}
```

- Per i dettagli sulle API, consulta la [ListJobs](#) guida di riferimento all'API AWS SDK for Rust.

StartCrawler

Il seguente esempio di codice mostra come utilizzare `StartCrawler`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

```

- Per i dettagli sulle API, consulta la [StartCrawler](#) guida di riferimento all'API AWS SDK for Rust.

StartJobRun

Il seguente esempio di codice mostra come utilizzare `StartJobRun`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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();
```

- Per i dettagli sulle API, consulta la [StartJobRun](#) guida di riferimento all'API AWS SDK for Rust.

Esempi di IAM che utilizzano SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l' AWS SDK per Rust con IAM.

Le nozioni di base sono esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Hello IAM

Gli esempi di codice seguenti mostrano come iniziare a utilizzare IAM.

SDK per Rust

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Da 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(())
}

```

Da 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);
            name
        })
        .collect();

    Ok(policy_names)
}
```

- Per i dettagli sull'API, consulta la guida di riferimento [ListPolicies](#) all'API AWS SDK for Rust.

Argomenti

- [Nozioni di base](#)
- [Operazioni](#)

Nozioni di base

Informazioni di base

Il seguente esempio di codice mostra come creare un utente e assumere un ruolo.

Warning

Per evitare rischi per la sicurezza, non utilizzare gli utenti IAM per l'autenticazione quando sviluppi software creato ad hoc o lavori con dati reali. Utilizza invece la federazione con un provider di identità come [AWS IAM Identity Center](#).

- Crea un utente che non disponga di autorizzazioni.
- Crea un ruolo che conceda l'autorizzazione per elencare i bucket Amazon S3 per l'account.

- Aggiungi una policy per consentire all'utente di assumere il ruolo.
- Assumi il ruolo ed elenca i bucket S3 utilizzando le credenziali temporanee, quindi ripulisci le risorse.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per informazioni dettagliate sulle API, consulta i seguenti argomenti nella Documentazione di riferimento delle API SDK AWS per Rust.
 - [AttachRolePolicy](#)
 - [CreateAccessKey](#)
 - [CreatePolicy](#)
 - [CreateRole](#)
 - [CreateUser](#)
 - [DeleteAccessKey](#)
 - [DeletePolicy](#)
 - [DeleteRole](#)
 - [DeleteUser](#)
 - [DeleteUserPolicy](#)
 - [DetachRolePolicy](#)
 - [PutUserPolicy](#)

Operazioni

AttachRolePolicy

Il seguente esempio di codice mostra come usare `AttachRolePolicy`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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
}
```

- Per i dettagli sulle API, consulta la [AttachRolePolicy](#) guida di riferimento all'API AWS SDK for Rust.

AttachUserPolicy

Il seguente esempio di codice mostra come utilizzare `AttachUserPolicy`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}

```

- Per i dettagli sulle API, consulta la [AttachUserPolicy](#) guida di riferimento all'API AWS SDK for Rust.

CreateAccessKey

Il seguente esempio di codice mostra come utilizzare `CreateAccessKey`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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())
}
```

- Per i dettagli sulle API, consulta la [CreateAccessKey](#) guida di riferimento all'API AWS SDK for Rust.

CreatePolicy

Il seguente esempio di codice mostra come utilizzare `CreatePolicy`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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())
}
```

- Per i dettagli sulle API, consulta la [CreatePolicy](#) guida di riferimento all'API AWS SDK for Rust.

CreateRole

Il seguente esempio di codice mostra come utilizzare `CreateRole`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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())
}
```

- Per i dettagli sulle API, consulta la [CreateRole](#) guida di riferimento all'API AWS SDK for Rust.

CreateServiceLinkedRole

Il seguente esempio di codice mostra come utilizzare `CreateServiceLinkedRole`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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)
}
```

- Per i dettagli sulle API, consulta la [CreateServiceLinkedRole](#) guida di riferimento all'API AWS SDK for Rust.

CreateUser

Il seguente esempio di codice mostra come utilizzare `CreateUser`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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())
}
```

- Per i dettagli sulle API, consulta la [CreateUser](#) guida di riferimento all'API AWS SDK for Rust.

DeleteAccessKey

Il seguente esempio di codice mostra come utilizzare `DeleteAccessKey`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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::<(), ()  
}
```

- Per i dettagli sulle API, consulta la [DeleteAccessKey](#) guida di riferimento all'API AWS SDK for Rust.

DeletePolicy

Il seguente esempio di codice mostra come utilizzare `DeletePolicy`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sulle API, consulta la [DeletePolicy](#) guida di riferimento all'API AWS SDK for Rust.

DeleteRole

Il seguente esempio di codice mostra come utilizzare `DeleteRole`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta la [DeleteRole](#) guida di riferimento all'API AWS SDK for Rust.

DeleteServiceLinkedRole

Il seguente esempio di codice mostra come utilizzare `DeleteServiceLinkedRole`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}

```

- Per i dettagli sulle API, consulta la [DeleteServiceLinkedRole](#) guida di riferimento all'API AWS SDK for Rust.

DeleteUser

Il seguente esempio di codice mostra come utilizzare `DeleteUser`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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
}

```

- Per i dettagli sulle API, consulta la [DeleteUser](#) guida di riferimento all'API AWS SDK for Rust.

DeleteUserPolicy

Il seguente esempio di codice mostra come utilizzare `DeleteUserPolicy`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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(())
}

```

```
}
```

- Per i dettagli sulle API, consulta la [DeleteUserPolicy](#) guida di riferimento all'API AWS SDK for Rust.

DetachRolePolicy

Il seguente esempio di codice mostra come utilizzare `DetachRolePolicy`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta la [DetachRolePolicy](#) guida di riferimento all'API AWS SDK for Rust.

DetachUserPolicy

Il seguente esempio di codice mostra come utilizzare `DetachUserPolicy`.

SDK per Rust

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta la [DetachUserPolicy](#) guida di riferimento all'API AWS SDK for Rust.

GetAccountPasswordPolicy

Il seguente esempio di codice mostra come utilizzare `GetAccountPasswordPolicy`.

SDK per Rust

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
pub async fn get_account_password_policy(
```

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

    Ok(response)
}
```

- Per i dettagli sulle API, consulta la [GetAccountPasswordPolicy](#) guida di riferimento all'API AWS SDK for Rust.

GetRole

Il seguente esempio di codice mostra come utilizzare `GetRole`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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)
}
```

- Per i dettagli sulle API, consulta la [GetRole](#) guida di riferimento all'API AWS SDK for Rust.

ListAttachedRolePolicies

Il seguente esempio di codice mostra come utilizzare `ListAttachedRolePolicies`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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)
}
```

- Per i dettagli sulle API, consulta la [ListAttachedRolePolicies](#) guida di riferimento all'API AWS SDK for Rust.

ListGroups

Il seguente esempio di codice mostra come utilizzare `ListGroups`.

SDK per Rust

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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)
}
```

- Per i dettagli sulle API, consulta la [ListGroups](#) guida di riferimento all'API AWS SDK for Rust.

ListPolicies

Il seguente esempio di codice mostra come utilizzare `ListPolicies`.

SDK per Rust

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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)
}

```

- Per i dettagli sulle API, consulta la [ListPolicies](#) guida di riferimento all'API AWS SDK for Rust.

ListRolePolicies

Il seguente esempio di codice mostra come utilizzare `ListRolePolicies`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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)
}
```

- Per i dettagli sulle API, consulta la [ListRolePolicies](#) guida di riferimento all'API AWS SDK for Rust.

ListRoles

Il seguente esempio di codice mostra come utilizzare `ListRoles`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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)
}
```

- Per i dettagli sulle API, consulta la [ListRoles](#) guida di riferimento all'API AWS SDK for Rust.

ListSAMLProviders

Il seguente esempio di codice mostra come utilizzare `ListSAMLProviders`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
pub async fn list_saml_providers(
    client: &Client,
) -> Result<ListSamlProvidersOutput, SdkError<ListSAMLProvidersError>> {
    let response = client.list_saml_providers().send().await?;

    Ok(response)
}
```

- Per i dettagli sull'API, consulta [List SAMLProviders](#) in AWS SDK for Rust API reference.

ListUsers

Il seguente esempio di codice mostra come utilizzare `ListUsers`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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)
}
```

- Per i dettagli sulle API, consulta la [ListUsers](#) guida di riferimento all'API AWS SDK for Rust.

AWS IoT esempi che utilizzano SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l' AWS SDK per Rust con. AWS IoT

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Operazioni](#)

Operazioni

DescribeEndpoint

Il seguente esempio di codice mostra come utilizzare `DescribeEndpoint`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta la [DescribeEndpoint](#) guida di riferimento all'API AWS SDK for Rust.

ListThings

Il seguente esempio di codice mostra come utilizzare `ListThings`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta la [ListThings](#) guida di riferimento all'API AWS SDK for Rust.

Esempi di Kinesis con SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l' AWS SDK per Rust con Kinesis.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Operazioni](#)
- [Esempi serverless](#)

Operazioni

CreateStream

Il seguente esempio di codice mostra come utilizzare `CreateStream`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta la [CreateStream](#) guida di riferimento all'API AWS SDK for Rust.

DeleteStream

Il seguente esempio di codice mostra come utilizzare `DeleteStream`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
async fn remove_stream(client: &Client, stream: &str) -> Result<(), Error> {
    client.delete_stream().stream_name(stream).send().await?;

    println!("Deleted stream.");

    Ok(())
}
```

- Per i dettagli sulle API, consulta la [DeleteStream](#) guida di riferimento all'API AWS SDK for Rust.

DescribeStream

Il seguente esempio di codice mostra come utilizzare `DescribeStream`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}

```

- Per i dettagli sulle API, consulta la [DescribeStream](#) guida di riferimento all'API AWS SDK for Rust.

ListStreams

Il seguente esempio di codice mostra come utilizzare `ListStreams`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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(())
}

```

- Per i dettagli sulle API, consulta la [ListStreams](#) guida di riferimento all'API AWS SDK for Rust.

PutRecord

Il seguente esempio di codice mostra come utilizzare `PutRecord`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta la [PutRecord](#) guida di riferimento all'API AWS SDK for Rust.

Esempi serverless

Richiamare una funzione Lambda da un trigger Kinesis

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un flusso Kinesis. La funzione recupera il payload Kinesis, lo decodifica da Base64 e registra il contenuto del record.

SDK per Rust

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Kinesis con Lambda tramite 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
}

```

Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Kinesis

Il seguente esempio di codice mostra come implementare una risposta batch parziale per le funzioni Lambda che ricevono eventi da un flusso Kinesis. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

SDK per Rust

Note

C'è di più su. [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite 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 esempi che utilizzano SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l' AWS SDK per Rust con. AWS KMS

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Operazioni](#)

Operazioni

CreateKey

Il seguente esempio di codice mostra come utilizzare `CreateKey`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta la [CreateKey](#) guida di riferimento all'API AWS SDK for Rust.

Decrypt

Il seguente esempio di codice mostra come utilizzare `Decrypt`.

SDK per Rust

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sull'API, [consulta Decrypt](#) in AWS SDK for Rust API reference.

Encrypt

Il seguente esempio di codice mostra come utilizzare. Encrypt

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sull'API, [consulta Encrypt](#) in AWS SDK for Rust API reference.

GenerateDataKey

Il seguente esempio di codice mostra come utilizzare `GenerateDataKey`

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta la [GenerateDataKey](#) guida di riferimento all'API AWS SDK for Rust.

GenerateDataKeyWithoutPlaintext

Il seguente esempio di codice mostra come utilizzare `GenerateDataKeyWithoutPlaintext`.

SDK per Rust

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta la [GenerateDataKeyWithoutPlaintext](#) guida di riferimento all'API AWS SDK for Rust.

GenerateRandom

Il seguente esempio di codice mostra come utilizzare `GenerateRandom`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta la [GenerateRandom](#) guida di riferimento all'API AWS SDK for Rust.

ListKeys

Il seguente esempio di codice mostra come utilizzare `ListKeys`.

SDK per Rust

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta la [ListKeys](#) guida di riferimento all'API AWS SDK for Rust.

ReEncrypt

Il seguente esempio di codice mostra come utilizzare `ReEncrypt`.

SDK per Rust

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta la [ReEncrypt](#) guida di riferimento all'API AWS SDK for Rust.

Esempi di Lambda con SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l' AWS SDK per Rust con Lambda.

Le nozioni di base sono esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Gli scenari sono esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

AWS i contributi della community sono esempi che sono stati creati e gestiti da più team. AWS Per fornire feedback, utilizza il meccanismo fornito negli archivi collegati.

Ogni esempio include un collegamento al codice sorgente completo, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Nozioni di base](#)
- [Operazioni](#)
- [Scenari](#)
- [Esempi serverless](#)
- [AWS contributi della comunità](#)

Nozioni di base

Informazioni di base

L'esempio di codice seguente mostra come:

- Crea un ruolo IAM e una funzione Lambda, quindi carica il codice del gestore.

- Richiamare la funzione con un singolo parametro e ottenere i risultati.
- Aggiorna il codice della funzione e configuralo con una variabile di ambiente.
- Richiamare la funzione con nuovi parametri e ottenere i risultati. Visualizza il log di esecuzione restituito.
- Elenca le funzioni dell'account, quindi elimina le risorse.

Per ulteriori informazioni sull'utilizzo di Lambda, consulta [Creare una funzione Lambda con la console](#).

SDK per Rust

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Il file Cargo.toml con dipendenze utilizzato in questo scenario.

```
[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"
```

Una raccolta di utilità che semplificano le invocazioni a Lambda per questo scenario. Questo file è `src/ations.rs` nella cassa.

```
use anyhow::anyhow;
use aws_sdk_iam::operation::{create_role::CreateRoleError,
    delete_role::DeleteRoleOutput};
use aws_sdk_lambda::{
    operation::{
        delete_function::DeleteFunctionOutput, get_function::GetFunctionOutput,
        invoke::InvokeOutput, list_functions::ListFunctionsOutput,
        update_function_code::UpdateFunctionCodeOutput,
        update_function_configuration::UpdateFunctionConfigurationOutput,
    },
    primitives::ByteStream,
    types::{Environment, FunctionCode, LastUpdateStatus, State},
};
use aws_sdk_s3::{
    error::ErrorMetadata,
    operation::{delete_bucket::DeleteBucketOutput,
        delete_object::DeleteObjectOutput},
    types::CreateBucketConfiguration,
};
use aws_smithy_types::Blob;
use serde::{ser::SerializeMap, Serialize};
use std::{fmt::Display, path::PathBuf, str::FromStr, time::Duration};
use tracing::{debug, info, warn};

/* Operation describes */
#[derive(Clone, Copy, Debug, Serialize)]
pub enum Operation {
    #[serde(rename = "plus")]
    Plus,
    #[serde(rename = "minus")]
    Minus,
    #[serde(rename = "times")]
    Times,
```

```

    #[serde(rename = "divided-by")]
    DividedBy,
}

impl FromStr for Operation {
    type Err = anyhow::Error;

    fn from_str(s: &str) -> Result<Self, Self::Err> {
        match s {
            "plus" => Ok(Operation::Plus),
            "minus" => Ok(Operation::Minus),
            "times" => Ok(Operation::Times),
            "divided-by" => Ok(Operation::DividedBy),
            _ => Err(anyhow!("Unknown operation {s}")),
        }
    }
}

impl Display for Operation {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        match self {
            Operation::Plus => write!(f, "plus"),
            Operation::Minus => write!(f, "minus"),
            Operation::Times => write!(f, "times"),
            Operation::DividedBy => write!(f, "divided-by"),
        }
    }
}

/**
 * InvokeArgs will be serialized as JSON and sent to the AWS Lambda handler.
 */
#[derive(Debug)]
pub enum InvokeArgs {
    Increment(i32),
    Arithmetic(Operation, i32, i32),
}

impl Serialize for InvokeArgs {
    fn serialize<S>(&self, serializer: S) -> Result<S::Ok, S::Error>
    where
        S: serde::Serializer,
    {
        match self {

```



```

        InvokeArgs::Increment(i) => serializer.serialize_i32(*i),
        InvokeArgs::Arithmetic(o, i, j) => {
            let mut map: S::SerializeMap = serializer.serialize_map(Some(3))?;
            map.serialize_key(&"op".to_string())?;
            map.serialize_value(&o.to_string())?;
            map.serialize_key(&"i".to_string())?;
            map.serialize_value(&i)?;
            map.serialize_key(&"j".to_string())?;
            map.serialize_value(&j)?;
            map.end()
        }
    }
}

/** A policy document allowing Lambda to execute this function on the account's
    behalf. */
const ROLE_POLICY_DOCUMENT: &str = r#{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": { "Service": "lambda.amazonaws.com" },
            "Action": "sts:AssumeRole"
        }
    ]
}";

/**
 * A LambdaManager gathers all the resources necessary to run the Lambda example
 * scenario.
 * This includes instantiated aws_sdk clients and details of resource names.
 */
pub struct LambdaManager {
    iam_client: aws_sdk_iam::Client,
    lambda_client: aws_sdk_lambda::Client,
    s3_client: aws_sdk_s3::Client,
    lambda_name: String,
    role_name: String,
    bucket: String,
    own_bucket: bool,
}

```

```

// These unit type structs provide nominal typing on top of String parameters for
LambdaManager::new
pub struct LambdaName(pub String);
pub struct RoleName(pub String);
pub struct Bucket(pub String);
pub struct OwnBucket(pub bool);

impl LambdaManager {
    pub fn new(
        iam_client: aws_sdk_iam::Client,
        lambda_client: aws_sdk_lambda::Client,
        s3_client: aws_sdk_s3::Client,
        lambda_name: LambdaName,
        role_name: RoleName,
        bucket: Bucket,
        own_bucket: OwnBucket,
    ) -> Self {
        Self {
            iam_client,
            lambda_client,
            s3_client,
            lambda_name: lambda_name.0,
            role_name: role_name.0,
            bucket: bucket.0,
            own_bucket: own_bucket.0,
        }
    }

    /**
     * Load the AWS configuration from the environment.
     * Look up lambda_name and bucket if none are given, or generate a random name
if not present in the environment.
     * If the bucket name is provided, the caller needs to have created the bucket.
     * If the bucket name is generated, it will be created.
     */
    pub async fn load_from_env(lambda_name: Option<String>, bucket: Option<String>)
-> Self {
        let sdk_config = aws_config::load_from_env().await;
        let lambda_name = LambdaName(lambda_name.unwrap_or_else(|| {
            std::env::var("LAMBDA_NAME").unwrap_or_else(|_|
"rust_lambda_example".to_string())
        }));
        let role_name = RoleName(format!("{}_role", lambda_name.0));
        let (bucket, own_bucket) =

```

```

        match bucket {
            Some(bucket) => (Bucket(bucket), false),
            None => (
                Bucket(std::env::var("LAMBDA_BUCKET").unwrap_or_else(|_| {
                    format!("rust-lambda-example-{}", uuid::Uuid::new_v4())
                })),
                true,
            ),
        };

let s3_client = aws_sdk_s3::Client::new(&sdk_config);

if own_bucket {
    info!("Creating bucket for demo: {}", bucket.0);
    s3_client
        .create_bucket()
        .bucket(bucket.0.clone())
        .create_bucket_configuration(
            CreateBucketConfiguration::builder()

.location_constraint(aws_sdk_s3::types::BucketLocationConstraint::from(
                sdk_config.region().unwrap().as_ref(),
            ))
            .build(),
        )
        .send()
        .await
        .unwrap();
}

Self::new(
    aws_sdk_iam::Client::new(&sdk_config),
    aws_sdk_lambda::Client::new(&sdk_config),
    s3_client,
    lambda_name,
    role_name,
    bucket,
    OwnBucket(own_bucket),
)
}

/**
 * Upload function code from a path to a zip file.
 * The zip file must have an AL2 Linux-compatible binary called `bootstrap`.

```

```

    * The easiest way to create such a zip is to use `cargo lambda build --output-
format Zip`.
    */
    async fn prepare_function(
        &self,
        zip_file: PathBuf,
        key: Option<String>,
    ) -> Result<FunctionCode, anyhow::Error> {
        let body = ByteStream::from_path(zip_file).await?;

        let key = key.unwrap_or_else(|| format!("{}_code", self.lambda_name));

        info!("Uploading function code to s3://{}/{}", self.bucket, key);
        let _ = self
            .s3_client
            .put_object()
            .bucket(self.bucket.clone())
            .key(key.clone())
            .body(body)
            .send()
            .await?;

        Ok(FunctionCode::builder()
            .s3_bucket(self.bucket.clone())
            .s3_key(key)
            .build())
    }

    /**
    * Create a function, uploading from a zip file.
    */
    pub async fn create_function(&self, zip_file: PathBuf) -> Result<String,
anyhow::Error> {
        let code = self.prepare_function(zip_file, None).await?;

        let key = code.s3_key().unwrap().to_string();

        let role = self.create_role().await.map_err(|e| anyhow!(e))?;

        info!("Created iam role, waiting 15s for it to become active");
        tokio::time::sleep(Duration::from_secs(15)).await;

        info!("Creating lambda function {}", self.lambda_name);
        let _ = self

```

```
        .lambda_client
        .create_function()
        .function_name(self.lambda_name.clone())
        .code(code)
        .role(role.arn())
        .runtime(aws_sdk_lambda::types::Runtime::Provided2)
        .handler("_unused")
        .send()
        .await
        .map_err( anyhow::Error::from )?;

    self.wait_for_function_ready().await?;

    self.lambda_client
        .publish_version()
        .function_name(self.lambda_name.clone())
        .send()
        .await?;

    Ok(key)
}

/**
 * Create an IAM execution role for the managed Lambda function.
 * If the role already exists, use that instead.
 */
async fn create_role(&self) -> Result<aws_sdk_iam::types::Role, CreateRoleError>
{
    info!("Creating execution role for function");
    let get_role = self
        .iam_client
        .get_role()
        .role_name(self.role_name.clone())
        .send()
        .await;
    if let Ok(get_role) = get_role {
        if let Some(role) = get_role.role {
            return Ok(role);
        }
    }

    let create_role = self
        .iam_client
        .create_role()
```

```

        .role_name(self.role_name.clone())
        .assume_role_policy_document(ROLE_POLICY_DOCUMENT)
        .send()
        .await;

    match create_role {
        Ok(create_role) => match create_role.role {
            Some(role) => Ok(role),
            None => Err(CreateRoleError::generic(
                ErrorMetadata::builder()
                    .message("CreateRole returned empty success")
                    .build(),
            )),
        },
        Err(err) => Err(err.into_service_error()),
    }
}

/**
 * Poll `is_function_ready` with a 1-second delay. It returns when the function
 * is ready or when there's an error checking the function's state.
 */
pub async fn wait_for_function_ready(&self) -> Result<(), anyhow::Error> {
    info!("Waiting for function");
    while !self.is_function_ready(None).await? {
        info!("Function is not ready, sleeping 1s");
        tokio::time::sleep(Duration::from_secs(1)).await;
    }
    Ok(())
}

/**
 * Check if a Lambda function is ready to be invoked.
 * A Lambda function is ready for this scenario when its state is active and its
 * LastUpdateStatus is Successful.
 * Additionally, if a sha256 is provided, the function must have that as its
 * current code hash.
 * Any missing properties or failed requests will be reported as an Err.
 */
async fn is_function_ready(
    &self,
    expected_code_sha256: Option<&str>,
) -> Result<bool, anyhow::Error> {
    match self.get_function().await {

```

```

Ok(func) => {
    if let Some(config) = func.configuration() {
        if let Some(state) = config.state() {
            info!(?state, "Checking if function is active");
            if !matches!(state, State::Active) {
                return Ok(false);
            }
        }
    }
    match config.last_update_status() {
        Some(last_update_status) => {
            info!(?last_update_status, "Checking if function is
ready");

            match last_update_status {
                LastUpdateStatus::Successful => {
                    // continue
                }
                LastUpdateStatus::Failed |
LastUpdateStatus::InProgress => {
                    return Ok(false);
                }
                unknown => {
                    warn!(
                        status_variant = unknown.as_str(),
                        "LastUpdateStatus unknown"
                    );
                    return Err(anyhow!(
                        "Unknown LastUpdateStatus, fn config is
{config:?}"
                    ));
                }
            }
        }
        None => {
            warn!("Missing last update status");
            return Ok(false);
        }
    };
    if expected_code_sha256.is_none() {
        return Ok(true);
    }
    if let Some(code_sha256) = config.code_sha256() {
        return Ok(code_sha256 ==
expected_code_sha256.unwrap_or_default());
    }
}

```

```
    }
  }
  Err(e) => {
    warn!(?e, "Could not get function while waiting");
  }
}
Ok(false)
}

/** Get the Lambda function with this Manager's name. */
pub async fn get_function(&self) -> Result<GetFunctionOutput, anyhow::Error> {
  info!("Getting lambda function");
  self.lambda_client
    .get_function()
    .function_name(self.lambda_name.clone())
    .send()
    .await
    .map_err(anyhow::Error::from)
}

/** List all Lambda functions in the current Region. */
pub async fn list_functions(&self) -> Result<ListFunctionsOutput, anyhow::Error>
{
  info!("Listing lambda functions");
  self.lambda_client
    .list_functions()
    .send()
    .await
    .map_err(anyhow::Error::from)
}

/** Invoke the lambda function using calculator InvokeArgs. */
pub async fn invoke(&self, args: InvokeArgs) -> Result<InvokeOutput,
anyhow::Error> {
  info!(?args, "Invoking {}", self.lambda_name);
  let payload = serde_json::to_string(&args)?;
  debug!(?payload, "Sending payload");
  self.lambda_client
    .invoke()
    .function_name(self.lambda_name.clone())
    .payload(Blob::new(payload))
    .send()
    .await
    .map_err(anyhow::Error::from)
}
```



```
}

/** Given a Path to a zip file, update the function's code and wait for the
update to finish. */
pub async fn update_function_code(
    &self,
    zip_file: PathBuf,
    key: String,
) -> Result<UpdateFunctionCodeOutput, anyhow::Error> {
    let function_code = self.prepare_function(zip_file, Some(key)).await?;

    info!("Updating code for {}", self.lambda_name);
    let update = self
        .lambda_client
        .update_function_code()
        .function_name(self.lambda_name.clone())
        .s3_bucket(self.bucket.clone())
        .s3_key(function_code.s3_key().unwrap().to_string())
        .send()
        .await
        .map_err(anyhow::Error::from)?;

    self.wait_for_function_ready().await?;

    Ok(update)
}

/** Update the environment for a function. */
pub async fn update_function_configuration(
    &self,
    environment: Environment,
) -> Result<UpdateFunctionConfigurationOutput, anyhow::Error> {
    info!(
        ?environment,
        "Updating environment for {}", self.lambda_name
    );
    let updated = self
        .lambda_client
        .update_function_configuration()
        .function_name(self.lambda_name.clone())
        .environment(environment)
        .send()
        .await
        .map_err(anyhow::Error::from)?;
```

```
        self.wait_for_function_ready().await?;

        Ok(updated)
    }

    /** Delete a function and its role, and if possible or necessary, its associated
code object and bucket. */
    pub async fn delete_function(
        &self,
        location: Option<String>,
    ) -> (
        Result<DeleteFunctionOutput, anyhow::Error>,
        Result<DeleteRoleOutput, anyhow::Error>,
        Option<Result<DeleteObjectOutput, anyhow::Error>>,
    ) {
        info!("Deleting lambda function {}", self.lambda_name);
        let delete_function = self
            .lambda_client
            .delete_function()
            .function_name(self.lambda_name.clone())
            .send()
            .await
            .map_err(anyhow::Error::from);

        info!("Deleting iam role {}", self.role_name);
        let delete_role = self
            .iam_client
            .delete_role()
            .role_name(self.role_name.clone())
            .send()
            .await
            .map_err(anyhow::Error::from);

        let delete_object: Option<Result<DeleteObjectOutput, anyhow::Error>> =
            if let Some(location) = location {
                info!("Deleting object {location}");
                Some(
                    self.s3_client
                        .delete_object()
                        .bucket(self.bucket.clone())
                        .key(location)
                        .send()
                        .await
                )
            } 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(),
        );
    }
}

```

Un file binario per eseguire lo scenario dall'inizio alla fine, utilizzando i flag della linea di comando per controllare alcuni comportamenti. Questo file è in `src/bin/scenario formato.rs` nella cassa.

```

/*
## 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");
}
}
```

- Per informazioni dettagliate sulle API, consulta i seguenti argomenti nella Documentazione di riferimento delle API SDK AWS per Rust.
 - [CreateFunction](#)
 - [DeleteFunction](#)
 - [GetFunction](#)
 - [Invoke](#)
 - [ListFunctions](#)
 - [UpdateFunctionCode](#)
 - [UpdateFunctionConfiguration](#)

Operazioni

CreateFunction

Il seguente esempio di codice mostra come usare `CreateFunction`

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/**
 * 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())
    }

```

- Per i dettagli sulle API, consulta la [CreateFunction](#) guida di riferimento all'API AWS SDK for Rust.

DeleteFunction

Il seguente esempio di codice mostra come utilizzare `DeleteFunction`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/** 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)
}

```

- Per i dettagli sulle API, consulta la [DeleteFunction](#) guida di riferimento all'API AWS SDK for Rust.

GetFunction

Il seguente esempio di codice mostra come utilizzare `GetFunction`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).


```
/** 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)
}
```

- Per i dettagli sulle API, consulta la [GetFunction](#) guida di riferimento all'API AWS SDK for Rust.

Invoke

Il seguente esempio di codice mostra come utilizzare `Invoke`.

SDK per Rust

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per informazioni dettagliate sull'API, consulta la pagina [Invoke](#) della documentazione di riferimento dell'API SDK AWS per Rust.

ListFunctions

Il seguente esempio di codice mostra come usare `ListFunctions`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/** 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)
}
```

- Per i dettagli sulle API, consulta la [ListFunctions](#) guida di riferimento all'API AWS SDK for Rust.

UpdateFunctionCode

Il seguente esempio di codice mostra come utilizzare `UpdateFunctionCode`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/** 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())
}

```

- Per i dettagli sulle API, consulta la [UpdateFunctionCode](#) guida di riferimento all'API AWS SDK for Rust.

UpdateFunctionConfiguration

Il seguente esempio di codice mostra come utilizzare `UpdateFunctionConfiguration`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/** 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)
}
```

- Per i dettagli sulle API, consulta la [UpdateFunctionConfiguration](#) guida di riferimento all'API AWS SDK for Rust.

Scenari

Creazione di un'applicazione serverless per gestire foto

Nell'esempio di codice seguente viene illustrato come creare un'applicazione serverless che consente agli utenti di gestire le foto mediante etichette.

SDK per Rust

Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su [GitHub](#).

Per approfondire l'origine di questo esempio, consulta il post su [AWS Community](#).

Servizi utilizzati in questo esempio

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

Esempi serverless

Connessione a un database Amazon RDS in una funzione Lambda

Il seguente esempio di codice mostra come implementare una funzione Lambda che si connette a un database RDS. La funzione effettua una semplice richiesta al database e restituisce il risultato.

SDK per Rust

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Connessione a un database Amazon RDS in una funzione Lambda tramite 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}")
    })))
}
```

Richiamare una funzione Lambda da un trigger Kinesis

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un flusso Kinesis. La funzione recupera il payload Kinesis, lo decodifica da Base64 e registra il contenuto del record.

SDK per Rust

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Kinesis con Lambda tramite 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
}
```

Richiamare una funzione Lambda da un trigger DynamoDB

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un flusso DynamoDB. La funzione recupera il payload DocumentDB e registra il contenuto del record.

SDK per Rust

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento DynamoDB con Lambda tramite 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(())
}

```

Richiamare una funzione Lambda da un trigger Amazon DocumentDB

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un flusso di modifiche di DocumentDB. La funzione recupera il payload DocumentDB e registra il contenuto del record.

SDK per Rust

Note

C'è di più su. [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Amazon DocumentDB con Lambda tramite Ruby.

```

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(())  
}
```

Invocare una funzione Lambda da un trigger Amazon MSK

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento generato dalla ricezione di record da un cluster Amazon MSK. La funzione recupera il payload MSK e registra il contenuto del record.

SDK per Rust

Note

C'è di più su. [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Consumo di un evento Amazon MSK con Lambda utilizzando Rust.

```
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
}
```

Richiamo di una funzione Lambda da un trigger Amazon S3

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dal caricamento di un oggetto in un bucket S3. La funzione recupera il nome del bucket S3 e la chiave dell'oggetto dal parametro evento e chiama l'API Amazon S3 per recuperare e registrare il tipo di contenuto dell'oggetto.

SDK per Rust

 Note

C'è di più su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento S3 con Lambda tramite 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(())
}
```

Richiamo di una funzione Lambda da un trigger Amazon SNS

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di messaggi da un argomento SNS. La funzione recupera i messaggi dal parametro dell'evento e registra il contenuto di ogni messaggio.

SDK per Rust

Note

C'è di più su. [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SNS con Lambda tramite 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
}
```



```
}

```

Richiamo di una funzione Lambda da un trigger Amazon SQS

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di messaggi da una coda SQS. La funzione recupera i messaggi dal parametro dell'evento e registra il contenuto di ogni messaggio.

SDK per Rust

Note

C'è di più su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SQS con Lambda tramite 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
}

```

Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Kinesis

Il seguente esempio di codice mostra come implementare una risposta batch parziale per le funzioni Lambda che ricevono eventi da un flusso Kinesis. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

SDK per Rust

Note

C'è di più su. [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite 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
}
```

Segnalazione di errori di elementi batch per funzioni Lambda con un trigger DynamoDB

Il seguente esempio di codice mostra come implementare una risposta batch parziale per le funzioni Lambda che ricevono eventi da un flusso DynamoDB. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

SDK per Rust

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite 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
}

```

Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Amazon SQS

Il seguente esempio di codice mostra come implementare una risposta batch parziale per le funzioni Lambda che ricevono eventi da una coda SQS. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

SDK per Rust

Note

C'è di più su [GitHub](#). Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di SQS con Lambda tramite 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_identifcier: record.message_id.unwrap(),
            }),
        }
    }

    Ok(SqsBatchResponse {
        batch_item_failures,
    })
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    run(service_fn(function_handler)).await
}
```

AWS contributi della comunità

Crea e testa un'applicazione serverless

Il seguente esempio di codice mostra come creare e testare un'applicazione serverless utilizzando API Gateway con Lambda e DynamoDB.

SDK per Rust

Mostra come creare e testare un'applicazione serverless composta da un API Gateway con Lambda e DynamoDB utilizzando Rust SDK.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. [GitHub](#)

Servizi utilizzati in questo esempio

- API Gateway
- DynamoDB
- Lambda

MediaLive esempi che utilizzano SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l' AWS SDK per Rust con. MediaLive

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Operazioni](#)

Operazioni

ListInputs

Il seguente esempio di codice mostra come utilizzare `ListInputs`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Elenca i nomi che hai MediaLive inserito e ARNs nella regione.

```
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(())
}
```

- Per i dettagli sulle API, consulta il riferimento [ListInputs](#) all'API AWS SDK for Rust.

MediaPackage esempi che utilizzano SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l' AWS SDK per Rust con. MediaPackage

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Operazioni](#)

Operazioni

ListChannels

Il seguente esempio di codice mostra come utilizzare `ListChannels`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Elenca i canali ARNs e le descrizioni.

```
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(())
}
```

- Per i dettagli sulle API, consulta la [ListChannels](#) guida di riferimento all'API AWS SDK for Rust.

ListOriginEndpoints

Il seguente esempio di codice mostra come utilizzare `ListOriginEndpoints`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Elenca le descrizioni degli endpoint e 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(())
}
```

- Per i dettagli sulle API, consulta il riferimento [ListOriginEndpoints](#) all'API AWS SDK for Rust.

Esempi di Amazon MSK con SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l' AWS SDK per Rust con Amazon MSK.

Ogni esempio include un collegamento al codice sorgente completo, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti


- [Esempi serverless](#)

Esempi serverless

Invocare una funzione Lambda da un trigger Amazon MSK

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento generato dalla ricezione di record da un cluster Amazon MSK. La funzione recupera il payload MSK e registra il contenuto del record.

SDK per Rust

 Note

C'è di più su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Consumo di un evento Amazon MSK con Lambda utilizzando Rust.

```
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  
}
```

Esempi di Amazon Polly con SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l' AWS SDK per Rust con Amazon Polly.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Gli scenari sono esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un collegamento al codice sorgente completo, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Operazioni](#)
- [Scenari](#)

Operazioni

DescribeVoices

Il seguente esempio di codice mostra come utilizzare `DescribeVoices`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta la [DescribeVoices](#) guida di riferimento all'API AWS SDK for Rust.

ListLexicons

Il seguente esempio di codice mostra come utilizzare `ListLexicons`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta la [ListLexicons](#) guida di riferimento all'API AWS SDK for Rust.

PutLexicon

Il seguente esempio di codice mostra come utilizzare `PutLexicon`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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(())
}

```

- Per i dettagli sulle API, consulta la [PutLexicon](#) guida di riferimento all'API AWS SDK for Rust.

SynthesizeSpeech

Il seguente esempio di codice mostra come utilizzare `SynthesizeSpeech`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta la [SynthesizeSpeech](#) guida di riferimento all'API AWS SDK for Rust.

Scenari

Conversione di sintesi vocale e di nuovo in testo

L'esempio di codice seguente mostra come:

- Utilizzare Amazon Polly per sintetizzare un file di input in testo normale (UTF-8) in un file audio.
- Carica il file audio in un bucket Amazon S3.
- Utilizzare Amazon Transcribe per convertire il file audio in testo.
- Visualizzare il testo.

SDK per Rust

Utilizza Amazon Polly per sintetizzare un file di input di testo normale (UTF-8) in un file audio, caricare il file audio in un bucket Amazon S3, utilizzare Amazon Transcribe per convertire il file audio in testo e visualizzare il testo.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su [GitHub](#).

Servizi utilizzati in questo esempio

- Amazon Polly
- Amazon S3
- Amazon Transcribe

Esempi di QLDB che utilizzano SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l' AWS SDK per Rust con QLDB.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, in cui è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Operazioni](#)

Operazioni

CreateLedger

Il seguente esempio di codice mostra come utilizzare `CreateLedger`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta la [CreateLedger](#) guida di riferimento all'API AWS SDK for Rust.

ListLedgers

Il seguente esempio di codice mostra come utilizzare `ListLedgers`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta la [ListLedgers](#) guida di riferimento all'API AWS SDK for Rust.

Esempi di Amazon RDS con SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l'AWS SDK per Rust con Amazon RDS.

Ogni esempio include un collegamento al codice sorgente completo, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Esempi serverless](#)

Esempi serverless

Connessione a un database Amazon RDS in una funzione Lambda

Il seguente esempio di codice mostra come implementare una funzione Lambda che si connette a un database RDS. La funzione effettua una semplice richiesta al database e restituisce il risultato.

SDK per Rust

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Connessione a un database Amazon RDS in una funzione Lambda tramite 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}")
    })))
}
```

Esempi di Amazon RDS Data Service con SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l' AWS SDK per Rust con Amazon RDS Data Service.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Operazioni](#)

Operazioni

ExecuteStatement

Il seguente esempio di codice mostra come utilizzare `ExecuteStatement`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta la [ExecuteStatement](#) guida di riferimento all'API AWS SDK for Rust.

Esempi di Amazon Rekognition con SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l' AWS SDK per Rust con Amazon Rekognition.

Gli scenari sono esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un collegamento al codice sorgente completo, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Scenari](#)

Scenari

Creazione di un'applicazione serverless per gestire foto

Nell'esempio di codice seguente viene illustrato come creare un'applicazione serverless che consente agli utenti di gestire le foto mediante etichette.

SDK per Rust

Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su [GitHub](#).

Per approfondire l'origine di questo esempio, consulta il post su [AWS Community](#).

Servizi utilizzati in questo esempio

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

Rilevamento di volti in un'immagine

L'esempio di codice seguente mostra come:

- Salva un'immagine in un bucket Amazon S3.
- Utilizza Amazon Rekognition per rilevare i dettagli del viso, come fascia di età, sesso ed emozione (ad esempio sorridente).
- Visualizzare questi dettagli.

SDK per Rust

Salva l'immagine in un bucket Amazon S3 con il prefisso uploads, utilizza Amazon Rekognition per rilevare i dettagli del viso, come fascia di età, sesso ed emozione (sorridente, ecc.) e visualizza tali dettagli.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su [GitHub](#).

Servizi utilizzati in questo esempio

- Amazon Rekognition
- Amazon S3

Salvataggio di EXIF e altre informazioni sull'immagine

L'esempio di codice seguente mostra come:

- Recuperare informazioni EXIF da un file JPG, JPEG o PNG.
- Carica il file immagine in un bucket Amazon S3.
- Utilizza Amazon Rekognition per identificare i tre attributi principali (etichette) nel file.
- Aggiungi le informazioni su EXIF ed etichette a una tabella Amazon DynamoDB nella regione.

SDK per Rust

Recupera le informazioni EXIF da un file JPG, JPEG o PNG, carica il file di immagine in un bucket Amazon S3, utilizza Amazon Rekognition per identificare i tre attributi principali (etichette in Amazon Rekognition) nel file e aggiungi le informazioni su EXIF ed etichette a una tabella Amazon DynamoDB nella regione.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su [GitHub](#).

Servizi utilizzati in questo esempio

- DynamoDB
- Amazon Rekognition
- Amazon S3

Esempi di Route 53 che utilizzano SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l' AWS SDK per Rust con Route 53.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Operazioni](#)

Operazioni

ListHostedZones

Il seguente esempio di codice mostra come utilizzare `ListHostedZones`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta la [ListHostedZones](#) guida di riferimento all'API AWS SDK for Rust.

Esempi di Amazon S3 con SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l' AWS SDK per Rust con Amazon S3.

Le nozioni di base sono esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Gli scenari sono esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un collegamento al codice sorgente completo, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Nozioni di base

Hello Amazon S3

Gli esempi di codice seguenti mostrano come iniziare a utilizzare Amazon S3.

SDK per Rust

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// 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 Utilità S3.

```

/// 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)
    }
}
```

- Per i dettagli sulle API, consulta la guida di riferimento [ListBuckets](#) all'API AWS SDK for Rust.

Argomenti

- [Nozioni di base](#)
- [Operazioni](#)
- [Scenari](#)
- [Esempi serverless](#)

Nozioni di base

Informazioni di base

L'esempio di codice seguente mostra come:

- Creare un bucket e caricare un file in tale bucket.
- Scaricare un oggetto da un bucket.
- Copiare un oggetto in una sottocartella in un bucket.

- Elencare gli oggetti in un bucket.
- Elimina il bucket e tutti gli oggetti in esso contenuti.

SDK per Rust

Note

C'è di più su. [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Codice per la crate binario che esegue lo scenario.

```
#![allow(clippy::result_large_err)]

//! Purpose
//! Shows how to use the AWS SDK for Rust to get started using
//! Amazon Simple Storage Service (Amazon S3). Create a bucket, move objects into
  and out of it,
  and delete all resources at the end of the demo.
  This example follows the steps in "Getting started with Amazon S3" in the
  Amazon S3
  user guide.
  - https://docs.aws.amazon.com/AmazonS3/latest/userguide/GetStartedWithS3.html

use aws_config::meta::region::RegionProviderChain;
use aws_sdk_s3::{config::Region, Client};
use s3_code_examples::error::S3ExampleError;
use uuid::Uuid;

#[tokio::main]
async fn main() -> Result<(), S3ExampleError> {
    let region_provider = RegionProviderChain::first_try(Region::new("us-west-2"));
    let region = region_provider.region().await.unwrap();
    let shared_config = aws_config::from_env().region(region_provider).load().await;
    let client = Client::new(&shared_config);
    let bucket_name = format!("amzn-s3-demo-bucket-{}", Uuid::new_v4());
    let file_name = "s3/testfile.txt".to_string();
    let key = "test file key name".to_string();
```

```

    let target_key = "target_key".to_string();

    if let Err(e) = run_s3_operations(region, client, bucket_name, file_name, key,
target_key).await
    {
        eprintln!("{:?}", e);
    };

    Ok(())
}

async fn run_s3_operations(
    region: Region,
    client: Client,
    bucket_name: String,
    file_name: String,
    key: String,
    target_key: String,
) -> Result<(), S3ExampleError> {
    s3_code_examples::create_bucket(&client, &bucket_name, &region).await?;
    let run_example: Result<(), S3ExampleError> = (async {
        s3_code_examples::upload_object(&client, &bucket_name, &file_name,
&key).await?;
        let _object = s3_code_examples::download_object(&client, &bucket_name,
&key).await;
        s3_code_examples::copy_object(&client, &bucket_name, &bucket_name, &key,
&target_key)
            .await?;
        s3_code_examples::list_objects(&client, &bucket_name).await?;
        s3_code_examples::clear_bucket(&client, &bucket_name).await?;
        Ok(())
    })
    .await;
    if let Err(err) = run_example {
        eprintln!("Failed to complete getting-started example: {err:?}");
    }
    s3_code_examples::delete_bucket(&client, &bucket_name).await?;

    Ok(())
}

```

Azioni comuni utilizzate dallo scenario.

```

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))
            }
        }
    }
}

```

```
}  
}
```

- Per informazioni dettagliate sulle API, consulta i seguenti argomenti nella Documentazione di riferimento delle API SDK AWS per Rust.
 - [CopyObject](#)
 - [CreateBucket](#)
 - [DeleteBucket](#)
 - [DeleteObjects](#)
 - [GetObject](#)
 - [ListObjectsV2](#)
 - [PutObject](#)

Operazioni

CompleteMultipartUpload

Il seguente esempio di codice mostra come utilizzare `CompleteMultipartUpload`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
// 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(),
    );
}

```

- Per i dettagli sulle API, consulta il riferimento [CompleteMultipartUpload](#) all'API AWS SDK for Rust.

CopyObject

Il seguente esempio di codice mostra come utilizzare `CopyObject`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// 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(())
}

```

- Per i dettagli sulle API, consulta il riferimento [CopyObject](#) all'API AWS SDK for Rust.

CreateBucket

Il seguente esempio di codice mostra come utilizzare `CreateBucket`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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))
    }
}))
}

```

- Per i dettagli sulle API, consulta il riferimento [CreateBucket](#) all'API AWS SDK for Rust.

CreateMultipartUpload

Il seguente esempio di codice mostra come utilizzare `CreateMultipartUpload`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

// 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?;

```

- Per i dettagli sulle API, consulta il riferimento [CreateMultipartUpload](#) all'API AWS SDK for Rust.

DeleteBucket

Il seguente esempio di codice mostra come utilizzare `DeleteBucket`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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))
        }
    }
}
}

```

- Per i dettagli sulle API, consulta il riferimento [DeleteBucket](#) all'API AWS SDK for Rust.

DeleteObject

Il seguente esempio di codice mostra come utilizzare `DeleteObject`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// 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(())
}

```

- Per i dettagli sulle API, consulta il riferimento [DeleteObject](#) all'API AWS SDK for Rust.

DeleteObjects

Il seguente esempio di codice mostra come utilizzare `DeleteObjects`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/// 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(())
}

```

- Per i dettagli sulle API, consulta il riferimento [DeleteObjects](#) all'API AWS SDK for Rust.

GetBucketLocation

Il seguente esempio di codice mostra come utilizzare `GetBucketLocation`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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(())
}


```

- Per i dettagli sulle API, consulta il riferimento [GetBucketLocation](#) all'API AWS SDK for Rust.

GetObject

Il seguente esempio di codice mostra come utilizzare `GetObject`.

SDK per Rust

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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)
}
```

```
}
```

- Per i dettagli sulle API, consulta il riferimento [GetObject](#) all'API AWS SDK for Rust.

ListBuckets

Il seguente esempio di codice mostra come utilizzare `ListBuckets`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}

```

- Per i dettagli sulle API, consulta il riferimento [ListBuckets](#) all'API AWS SDK for Rust.

ListObjectVersions

Il seguente esempio di codice mostra come utilizzare `ListObjectVersions`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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(())
}
```

- Per i dettagli sulle API, consulta il riferimento [ListObjectVersions](#) all'API AWS SDK for Rust.

ListObjectsV2

Il seguente esempio di codice mostra come utilizzare `ListObjectsV2`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sull'API, consulta la [ListObjectsversione 2](#) in AWS SDK for Rust API reference.

PutObject

Il seguente esempio di codice mostra come utilizzare. PutObject

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).


```
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)
}
```

- Per i dettagli sulle API, consulta il riferimento [PutObject](#) all'API AWS SDK for Rust.

UploadPart

Il seguente esempio di codice mostra come utilizzare UploadPart.

SDK per Rust

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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?;
```

- Per i dettagli sulle API, consulta il riferimento [UploadPart](#) all'API AWS SDK for Rust.

Scenari

Conversione di sintesi vocale e di nuovo in testo

L'esempio di codice seguente mostra come:

- Utilizzare Amazon Polly per sintetizzare un file di input in testo normale (UTF-8) in un file audio.
- Carica il file audio in un bucket Amazon S3.
- Utilizzare Amazon Transcribe per convertire il file audio in testo.
- Visualizzare il testo.

SDK per Rust

Utilizza Amazon Polly per sintetizzare un file di input di testo normale (UTF-8) in un file audio, caricare il file audio in un bucket Amazon S3, utilizzare Amazon Transcribe per convertire il file audio in testo e visualizzare il testo.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su [GitHub](#).

Servizi utilizzati in questo esempio

- Amazon Polly
- Amazon S3
- Amazon Transcribe

Creazione di un URL prefirmato

Il seguente esempio di codice mostra come creare un URL predefinito per Amazon S3 e caricare un oggetto.

SDK per Rust

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Crea richieste di preassegnazione per oggetti GET S3.

```
/// 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(())
}

```

Crea richieste di preassegnazione agli oggetti PUT S3.

```

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())  
}
```

Creazione di un'applicazione serverless per gestire foto

Nell'esempio di codice seguente viene illustrato come creare un'applicazione serverless che consente agli utenti di gestire le foto mediante etichette.

SDK per Rust

Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guardate l'esempio completo su. [GitHub](#)

Per approfondire l'origine di questo esempio, consulta il post su [AWS Community](#).

Servizi utilizzati in questo esempio

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

Rilevamento di volti in un'immagine

L'esempio di codice seguente mostra come:

- Salva un'immagine in un bucket Amazon S3.
- Utilizza Amazon Rekognition per rilevare i dettagli del viso, come fascia di età, sesso ed emozione (ad esempio sorridente).
- Visualizzare questi dettagli.

SDK per Rust

Salva l'immagine in un bucket Amazon S3 con il prefisso uploads, utilizza Amazon Rekognition per rilevare i dettagli del viso, come fascia di età, sesso ed emozione (sorridente, ecc.) e visualizza tali dettagli.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su [GitHub](#).

Servizi utilizzati in questo esempio

- Amazon Rekognition
- Amazon S3

Recuperare un oggetto da un bucket se è stato modificato

L'esempio di codice seguente mostra come leggere i dati da un oggetto in un bucket S3, ma solo se il bucket non è stato modificato dall'ultimo recupero.

SDK per Rust

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta il riferimento [GetObject](#) all'API AWS SDK for Rust.

Salvataggio di EXIF e altre informazioni sull'immagine

L'esempio di codice seguente mostra come:

- Recuperare informazioni EXIF da un file JPG, JPEG o PNG.
- Carica il file immagine in un bucket Amazon S3.
- Utilizza Amazon Rekognition per identificare i tre attributi principali (etichette) nel file.
- Aggiungi le informazioni su EXIF ed etichette a una tabella Amazon DynamoDB nella regione.

SDK per Rust

Recupera le informazioni EXIF da un file JPG, JPEG o PNG, carica il file di immagine in un bucket Amazon S3, utilizza Amazon Rekognition per identificare i tre attributi principali (etichette in Amazon Rekognition) nel file e aggiungi le informazioni su EXIF ed etichette a una tabella Amazon DynamoDB nella regione.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su [GitHub](#).

Servizi utilizzati in questo esempio

- DynamoDB
- Amazon Rekognition
- Amazon S3

Test di unità e integrazione con un SDK

Il seguente esempio di codice mostra come utilizzare esempi di tecniche ottimali per la scrittura di test unitari e di integrazione utilizzando un AWS SDK.

SDK per Rust

Note

C'è di più su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Cargo.toml per esempi di test.

```
[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"

```

Esempio di test di unità utilizzando automock e un wrapper di servizi.

```

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);
}
}

```

Esempio di test di integrazione utilizzando `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(&[]);
}
}

```


Caricamento o download di file di grandi dimensioni

Il seguente esempio di codice mostra come caricare o scaricare file di grandi dimensioni da e verso Amazon S3.

Per ulteriori informazioni, consulta [Caricamento di un oggetto utilizzando il caricamento in più parti](#).

SDK per Rust

Note

C'è altro su GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

Esempi serverless

Richiamo di una funzione Lambda da un trigger Amazon S3

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dal caricamento di un oggetto in un bucket S3. La funzione recupera il nome del bucket S3 e la chiave dell'oggetto dal parametro evento e chiama l'API Amazon S3 per recuperare e registrare il tipo di contenuto dell'oggetto.

SDK per Rust

Note

C'è di più su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento S3 con Lambda tramite 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 Esempi di intelligenza artificiale che utilizzano SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l' AWS SDK per Rust con SageMaker AI.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Operazioni](#)

Operazioni

ListNotebookInstances

Il seguente esempio di codice mostra come utilizzare `ListNotebookInstances`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta il riferimento [ListNotebookInstances](#) all'API AWS SDK for Rust.

ListTrainingJobs

Il seguente esempio di codice mostra come utilizzare `ListTrainingJobs`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}

```

- Per i dettagli sulle API, consulta il riferimento [ListTrainingJobs](#) all'API AWS SDK for Rust.

Esempi di Secrets Manager con SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l' AWS SDK per Rust con Secrets Manager.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Operazioni](#)

Operazioni

GetSecretValue

Il seguente esempio di codice mostra come utilizzare `GetSecretValue`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta il riferimento [GetSecretValue](#) all'API AWS SDK for Rust.

Esempi di API Amazon SES v2 con SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l'AWS SDK per Rust con l'API Amazon SES v2.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Gli scenari sono esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un collegamento al codice sorgente completo, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Operazioni](#)
- [Scenari](#)

Operazioni

CreateContact

Il seguente esempio di codice mostra come utilizzare `CreateContact`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta il riferimento [CreateContact](#) all'API AWS SDK for Rust.

CreateContactList

Il seguente esempio di codice mostra come utilizzare `CreateContactList`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta il riferimento [CreateContactList](#) all'API AWS SDK for Rust.

CreateEmailIdentity

Il seguente esempio di codice mostra come utilizzare `CreateEmailIdentity`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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) ),
        },
    }
}

```

- Per i dettagli sulle API, consulta il riferimento [CreateEmailIdentity](#) all'API AWS SDK for Rust.

CreateEmailTemplate

Il seguente esempio di codice mostra come utilizzare `CreateEmailTemplate`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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)),
        },
    }
}

```

- Per i dettagli sulle API, consulta il riferimento [CreateEmailTemplate](#) all'API AWS SDK for Rust.

DeleteContactList

Il seguente esempio di codice mostra come utilizzare `DeleteContactList`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

    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}")),
    }

```

- Per i dettagli sulle API, consulta il riferimento [DeleteContactList](#) all'API AWS SDK for Rust.

DeleteEmailIdentity

Il seguente esempio di codice mostra come utilizzare `DeleteEmailIdentity`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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));
        }
    }
}

```

- Per i dettagli sulle API, consulta il riferimento [DeleteEmailIdentity](#) all'API AWS SDK for Rust.

DeleteEmailTemplate

Il seguente esempio di codice mostra come utilizzare `DeleteEmailTemplate`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

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

- Per i dettagli sulle API, consulta il riferimento [DeleteEmailTemplate](#) all'API AWS SDK for Rust.

GetEmailIdentity

Il seguente esempio di codice mostra come utilizzare `GetEmailIdentity`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Determina se un indirizzo e-mail è stato verificato.

```
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(())
}
```

- Per i dettagli sulle API, consulta il riferimento [GetEmailIdentity](#) all'API AWS SDK for Rust.

ListContactLists

Il seguente esempio di codice mostra come utilizzare `ListContactLists`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta il riferimento [ListContactLists](#) all'API AWS SDK for Rust.

ListContacts

Il seguente esempio di codice mostra come utilizzare `ListContacts`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta il riferimento [ListContacts](#) all'API AWS SDK for Rust.

SendEmail

Il seguente esempio di codice mostra come utilizzare `SendEmail`.

SDK per Rust

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Invia un messaggio a tutti i membri dell'elenco di contatti.

```
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(())
}

```

Invia un messaggio a tutti i membri dell'elenco dei contatti utilizzando un modello.

```

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)),
    }

```

- Per i dettagli sull'API, consulta la [SendEmail](#) guida di riferimento all'API AWS SDK for Rust.

Scenari

Scenario della newsletter

Il seguente esempio di codice mostra come eseguire lo scenario di newsletter Amazon SES API v2.

SDK per Rust

Note

C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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}"));
    }
}
```

- Per informazioni dettagliate sulle API, consulta i seguenti argomenti nella Documentazione di riferimento delle API SDK AWS per Rust.
 - [CreateContact](#)
 - [CreateContactList](#)
 - [CreateEmailIdentity](#)
 - [CreateEmailTemplate](#)
 - [DeleteContactList](#)
 - [DeleteEmailIdentity](#)
 - [DeleteEmailTemplate](#)
 - [ListContacts](#)
 - [SendEmail.semplice](#)
 - [SendEmail.modello](#)

Esempi di Amazon SNS con SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l' AWS SDK per Rust con Amazon SNS.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Gli scenari sono esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un collegamento al codice sorgente completo, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Operazioni](#)
- [Scenari](#)
- [Esempi serverless](#)

Operazioni

CreateTopic

Il seguente esempio di codice mostra come utilizzare `CreateTopic`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta il riferimento [CreateTopic](#) all'API AWS SDK for Rust.

ListTopics

Il seguente esempio di codice mostra come utilizzare `ListTopics`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta il riferimento [ListTopics](#) all'API AWS SDK for Rust.

Publish

Il seguente esempio di codice mostra come utilizzare `Publish`.

SDK per Rust

 Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per informazioni dettagliate sulle API, consulta [Pubblicazione](#) nella Documentazione di riferimento degli SDK AWS per l'API Rust.

Subscribe

Il seguente esempio di codice mostra come usare `Subscribe`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Sottoscrivi un indirizzo email a un argomento.

```
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(())
}
```

- Per informazioni dettagliate sulle API, consulta [Sottoscrizione](#) nella Documentazione di riferimento degli SDK AWS per l'API Rust.

Scenari

Creazione di un'applicazione serverless per gestire foto

Nell'esempio di codice seguente viene illustrato come creare un'applicazione serverless che consente agli utenti di gestire le foto mediante etichette.

SDK per Rust

Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su [GitHub](#).

Per approfondire l'origine di questo esempio, consulta il post su [AWS Community](#).

Servizi utilizzati in questo esempio


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

Esempi serverless

Richiamo di una funzione Lambda da un trigger Amazon SNS

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di messaggi da un argomento SNS. La funzione recupera i messaggi dal parametro dell'evento e registra il contenuto di ogni messaggio.

SDK per Rust

 Note

C'è di più su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SNS con Lambda tramite 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
}
```

Esempi di Amazon SQS con SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l' AWS SDK per Rust con Amazon SQS.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Operazioni](#)
- [Esempi serverless](#)

Operazioni

ListQueues

Il seguente esempio di codice mostra come utilizzare `ListQueues`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Recupera la prima coda Amazon SQS elencata nella regione.

```
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())
}
```

- Per i dettagli sulle API, consulta il riferimento [ListQueues](#) all'API AWS SDK for Rust.

ReceiveMessage

Il seguente esempio di codice mostra come utilizzare `ReceiveMessage`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta il riferimento [ReceiveMessage](#) all'API AWS SDK for Rust.

SendMessage

Il seguente esempio di codice mostra come utilizzare `SendMessage`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```


- Per i dettagli sulle API, consulta il riferimento [SendMessage](#) all'API AWS SDK for Rust.

Esempi serverless

Richiamo di una funzione Lambda da un trigger Amazon SQS

Il seguente esempio di codice mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di messaggi da una coda SQS. La funzione recupera i messaggi dal parametro dell'evento e registra il contenuto di ogni messaggio.

SDK per Rust

 Note

C'è di più su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SQS con Lambda tramite 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
}
```

Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Amazon SQS

Il seguente esempio di codice mostra come implementare una risposta batch parziale per le funzioni Lambda che ricevono eventi da una coda SQS. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

SDK per Rust

Note

C'è di più su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di SQS con Lambda tramite Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::{
    event::sqs::{SqsBatchResponse, SqsEvent},
    sqs::{BatchItemFailure, SqsMessage},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn process_record(_: &SqsMessage) -> Result<(), Error> {
    Err(Error::from("Error processing message"))
}

async fn function_handler(event: LambdaEvent<SqsEvent>) -> Result<SqsBatchResponse,
Error> {
    let mut batch_item_failures = Vec::new();
    for record in event.payload.records {
        match process_record(&record).await {
            Ok(_) => (),
            Err(_) => batch_item_failures.push(BatchItemFailure {
                item_identifier: record.message_id.unwrap(),
            }),
        }
    }

    Ok(SqsBatchResponse {
        batch_item_failures,
```

```
    })  
  }  
  
  #[tokio::main]  
  async fn main() -> Result<(), Error> {  
    run(service_fn(function_handler)).await  
  }  
}
```

AWS STS esempi che utilizzano SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l' AWS SDK per Rust con. AWS STS

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Operazioni](#)

Operazioni

AssumeRole

Il seguente esempio di codice mostra come utilizzare `AssumeRole`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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),
    }
}

```

- Per i dettagli sulle API, consulta il riferimento [AssumeRole](#) all'API AWS SDK for Rust.

Esempi di Systems Manager che utilizzano SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l' AWS SDK per Rust con Systems Manager.

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un collegamento al codice sorgente completo, dove è possibile trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Operazioni](#)

Operazioni

DescribeParameters

Il seguente esempio di codice mostra come utilizzare `DescribeParameters`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta il riferimento [DescribeParameters](#) all'API AWS SDK for Rust.

GetParameter

Il seguente esempio di codice mostra come utilizzare `GetParameter`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).


```

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)
}

```

- Per i dettagli sulle API, consulta il riferimento [GetParameter](#) all'API AWS SDK for Rust.

PutParameter

Il seguente esempio di codice mostra come utilizzare `PutParameter`.

SDK per Rust

Note

C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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(())
}
```

- Per i dettagli sulle API, consulta il riferimento [PutParameter](#) all'API AWS SDK for Rust.

Esempi di Amazon Transcribe con SDK per Rust

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando l' AWS SDK per Rust con Amazon Transcribe.

Gli scenari sono esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un collegamento al codice sorgente completo, dove puoi trovare istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- [Scenari](#)

Scenari

Conversione di sintesi vocale e di nuovo in testo

L'esempio di codice seguente mostra come:

- Utilizzare Amazon Polly per sintetizzare un file di input in testo normale (UTF-8) in un file audio.
- Carica il file audio in un bucket Amazon S3.
- Utilizzare Amazon Transcribe per convertire il file audio in testo.

- Visualizzare il testo.

SDK per Rust

Utilizza Amazon Polly per sintetizzare un file di input di testo normale (UTF-8) in un file audio, caricare il file audio in un bucket Amazon S3, utilizzare Amazon Transcribe per convertire il file audio in testo e visualizzare il testo.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su [GitHub](#).

Servizi utilizzati in questo esempio

- Amazon Polly
- Amazon S3
- Amazon Transcribe

Sicurezza per questo AWS prodotto o servizio

La sicurezza cloud di Amazon Web Services (AWS) è la priorità più alta. In quanto cliente AWS, è possibile trarre vantaggio da un'architettura di data center e di rete progettata per soddisfare i requisiti delle organizzazioni più esigenti a livello di sicurezza. La sicurezza è una responsabilità condivisa tra AWS e te. Il [modello di responsabilità condivisa](#) descrive questo come sicurezza del cloud e sicurezza nel cloud.

Security of the Cloud: AWS è responsabile della protezione dell'infrastruttura che gestisce tutti i servizi offerti nel AWS Cloud e della fornitura di servizi che è possibile utilizzare in modo sicuro. La nostra responsabilità in AWS materia di sicurezza è la massima priorità e l'efficacia della nostra sicurezza viene regolarmente testata e verificata da revisori di terze parti nell'ambito dei Programmi di [AWS conformità](#).

Sicurezza nel cloud: la responsabilità dell'utente è determinata dal AWS servizio utilizzato e da altri fattori, tra cui la sensibilità dei dati, i requisiti dell'organizzazione e le leggi e i regolamenti applicabili.

Questo AWS prodotto o servizio segue il [modello di responsabilità condivisa](#) attraverso i servizi specifici di Amazon Web Services (AWS) che supporta. Per informazioni sulla sicurezza dei AWS servizi, consulta la [pagina della documentazione sulla sicurezza del AWS servizio](#) e [AWS i servizi che rientrano nell'ambito delle iniziative di AWS conformità previste dal programma di conformità](#).

Argomenti

- [Protezione dei dati in questo AWS prodotto o servizio](#)
- [Convalida della conformità per questo AWS prodotto o servizio](#)
- [Sicurezza dell'infrastruttura per questo AWS prodotto o servizio](#)
- [Implica una versione TLS minima nel AWS SDK for Rust](#)

Protezione dei dati in questo AWS prodotto o servizio

Il [modello di responsabilità AWS condivisa](#) si applica alla protezione dei dati in questo AWS prodotto o servizio. Come descritto in questo modello, AWS è responsabile della protezione dell'infrastruttura globale che gestisce tutti i Cloud AWS. L'utente è responsabile del controllo dei contenuti ospitati su questa infrastruttura. L'utente è inoltre responsabile della configurazione della protezione e delle attività di gestione per i Servizi AWS utilizzati. Per ulteriori informazioni sulla

privacy dei dati, vedi le [Domande frequenti sulla privacy dei dati](#). Per informazioni sulla protezione dei dati in Europa, consulta il post del blog relativo al [Modello di responsabilità condivisa AWS e GDPR](#) nel Blog sulla sicurezza AWS .

Ai fini della protezione dei dati, consigliamo di proteggere Account AWS le credenziali e configurare i singoli utenti con AWS IAM Identity Center or AWS Identity and Access Management (IAM). In tal modo, a ogni utente verranno assegnate solo le autorizzazioni necessarie per svolgere i suoi compiti. Ti suggeriamo, inoltre, di proteggere i dati nei seguenti modi:

- Utilizza l'autenticazione a più fattori (MFA) con ogni account.
- Usa SSL/TLS per comunicare con le risorse. AWS È richiesto TLS 1.2 ed è consigliato TLS 1.3.
- Configura l'API e la registrazione delle attività degli utenti con. AWS CloudTrail Per informazioni sull'utilizzo dei CloudTrail percorsi per acquisire AWS le attività, consulta [Lavorare con i CloudTrail percorsi](#) nella Guida per l'AWS CloudTrail utente.
- Utilizza soluzioni di AWS crittografia, insieme a tutti i controlli di sicurezza predefiniti all'interno Servizi AWS.
- Utilizza i servizi di sicurezza gestiti avanzati, come Amazon Macie, che aiutano a individuare e proteggere i dati sensibili archiviati in Amazon S3.
- Se hai bisogno di moduli crittografici convalidati FIPS 140-3 per accedere AWS tramite un'interfaccia a riga di comando o un'API, usa un endpoint FIPS. Per ulteriori informazioni sugli endpoint FIPS disponibili, consulta il [Federal Information Processing Standard \(FIPS\) 140-3](#).

Ti consigliamo di non inserire mai informazioni riservate o sensibili, ad esempio gli indirizzi e-mail dei clienti, nei tag o nei campi di testo in formato libero, ad esempio nel campo Nome. Ciò include quando si lavora con questo AWS prodotto o servizio o altro Servizi AWS utilizzando la console, l'API o. AWS CLI AWS SDKs I dati inseriti nei tag o nei campi di testo in formato libero utilizzati per i nomi possono essere utilizzati per i la fatturazione o i log di diagnostica. Quando fornisci un URL a un server esterno, ti suggeriamo vivamente di non includere informazioni sulle credenziali nell'URL per convalidare la tua richiesta al server.

Convalida della conformità per questo AWS prodotto o servizio

Per sapere se un Servizio AWS programma rientra nell'ambito di specifici programmi di conformità, consulta Servizi AWS la sezione [Scope by Compliance Program Servizi AWS](#) e scegli il programma di conformità che ti interessa. Per informazioni generali, consulta Programmi di [AWS conformità Programmi](#) di di .

È possibile scaricare report di audit di terze parti utilizzando AWS Artifact. Per ulteriori informazioni, consulta [Scaricamento dei report in AWS Artifact](#).

La vostra responsabilità di conformità durante l'utilizzo Servizi AWS è determinata dalla sensibilità dei dati, dagli obiettivi di conformità dell'azienda e dalle leggi e dai regolamenti applicabili. AWS fornisce le seguenti risorse per contribuire alla conformità:

- [Governance e conformità per la sicurezza](#): queste guide all'implementazione di soluzioni illustrano considerazioni relative all'architettura e i passaggi per implementare le funzionalità di sicurezza e conformità.
- [Riferimento ai servizi idonei alla normativa HIPAA: elenca i servizi](#) idonei alla normativa HIPAA. Non tutti Servizi AWS sono idonei all'HIPAA.
- [AWS Risorse per la per la conformità](#): questa raccolta di cartelle di lavoro e guide potrebbe essere valida per il tuo settore e la tua località.
- [AWS Guide alla conformità dei clienti](#): comprendi il modello di responsabilità condivisa attraverso la lente della conformità. Le guide riassumono le migliori pratiche per la protezione Servizi AWS e mappano le linee guida per i controlli di sicurezza su più framework (tra cui il National Institute of Standards and Technology (NIST), il Payment Card Industry Security Standards Council (PCI) e l'International Organization for Standardization (ISO)).
- [Evaluating Resources with Rules](#) nella AWS Config Developer Guide: il AWS Config servizio valuta la conformità delle configurazioni delle risorse alle pratiche interne, alle linee guida e alle normative del settore.
- [AWS Security Hub](#)— Ciò Servizio AWS fornisce una visione completa dello stato di sicurezza interno. AWS La Centrale di sicurezza utilizza i controlli di sicurezza per valutare le risorse AWS e verificare la conformità agli standard e alle best practice del settore della sicurezza. Per un elenco dei servizi e dei controlli supportati, consulta la pagina [Documentazione di riferimento sui controlli della Centrale di sicurezza](#).
- [Amazon GuardDuty](#): Servizio AWS rileva potenziali minacce ai tuoi carichi di lavoro Account AWS, ai contenitori e ai dati monitorando l'ambiente alla ricerca di attività sospette e dannose. GuardDuty può aiutarti a soddisfare vari requisiti di conformità, come lo standard PCI DSS, soddisfacendo i requisiti di rilevamento delle intrusioni imposti da determinati framework di conformità.
- [AWS Audit Manager](#)— Ciò Servizio AWS consente di verificare continuamente l' AWS utilizzo per semplificare la gestione del rischio e la conformità alle normative e agli standard di settore.

Questo AWS prodotto o servizio segue il [modello di responsabilità condivisa](#) attraverso i servizi specifici di Amazon Web Services (AWS) che supporta. Per informazioni sulla sicurezza dei AWS

servizi, consulta la [pagina della documentazione sulla sicurezza del AWS servizio](#) e [AWS i servizi che rientrano nell'ambito delle iniziative di AWS conformità previste dal programma di conformità](#).

Sicurezza dell'infrastruttura per questo AWS prodotto o servizio

Questo AWS prodotto o servizio utilizza servizi gestiti ed è pertanto protetto dalla sicurezza di rete AWS globale. Per informazioni sui servizi AWS di sicurezza e su come AWS protegge l'infrastruttura, consulta [AWS Cloud Security](#). Per progettare il tuo AWS ambiente utilizzando le migliori pratiche per la sicurezza dell'infrastruttura, vedi [Infrastructure Protection](#) in Security Pillar AWS Well-Architected Framework.

Utilizzate chiamate API AWS pubblicate per accedere a questo AWS Prodotto o Servizio attraverso la rete. I client devono supportare quanto segue:

- Transport Layer Security (TLS). È richiesto TLS 1.2 ed è consigliato TLS 1.3.
- Suite di cifratura con Perfect Forward Secrecy (PFS), ad esempio Ephemeral Diffie-Hellman (DHE) o Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). La maggior parte dei sistemi moderni, come Java 7 e versioni successive, supporta tali modalità.

Inoltre, le richieste devono essere firmate utilizzando un ID chiave di accesso e una chiave di accesso segreta associata a un principale IAM. O puoi utilizzare [AWS Security Token Service](#) (AWS STS) per generare credenziali di sicurezza temporanee per sottoscrivere le richieste.

Questo AWS prodotto o servizio segue il [modello di responsabilità condivisa](#) attraverso i servizi specifici di Amazon Web Services (AWS) che supporta. Per informazioni sulla sicurezza dei AWS servizi, consulta la [pagina della documentazione sulla sicurezza del AWS servizio](#) e [AWS i servizi che rientrano nell'ambito delle iniziative di AWS conformità previste dal programma di conformità](#).

Implica una versione TLS minima nel AWS SDK for Rust

AWS SDK for Rust Utilizza TLS per aumentare la sicurezza durante la comunicazione con i servizi. AWS Per impostazione predefinita, l'SDK applica una versione TLS minima di 1.2. Per impostazione predefinita, l'SDK negozia anche la versione più alta di TLS disponibile sia per l'applicazione client che per il servizio. Ad esempio, l'SDK potrebbe essere in grado di negoziare TLS 1.3.

Una particolare versione TLS può essere applicata nell'applicazione fornendo la configurazione manuale del connettore TCP utilizzato dall'SDK. Per illustrare ciò, l'esempio seguente mostra come applicare TLS 1.3.

Note

Alcuni AWS servizi non supportano ancora TLS 1.3, quindi l'applicazione di questa versione potrebbe influire sull'interoperabilità dell'SDK. Consigliamo di testare questa configurazione con ogni servizio prima della distribuzione in produzione.

```
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(())
}
```

Appendice: casse Rust utilizzate dall'SDK

Questa appendice contiene informazioni avanzate sulle casse utilizzate da AWS SDK for Rust. Ciò include i componenti Smithy che utilizza, le casse che potrebbe essere necessario utilizzare in determinate circostanze di costruzione e altre informazioni.

casse Smithy

Il AWS SDK for Rust è basato su [Smithy](#), come la maggior parte dei AWS SDKs. Smithy è un linguaggio usato per descrivere i tipi di dati e le funzioni offerte dall'SDK. Questi modelli vengono quindi utilizzati per aiutare a creare l'SDK stesso.

[Quando si esaminano le versioni dell'SDK per le casse Rust e quelle delle sue dipendenze Smithy, potrebbe essere utile sapere che tutte queste casse utilizzano la numerazione semantica delle versioni standard.](#)

[Per ulteriori informazioni dettagliate sulle casse Smithy per Rust, consulta Smithy Rust Design.](#)

Casse utilizzate con l'SDK per Rust

Esistono diverse casse Smithy pubblicate da AWS. Alcuni di questi sono rilevanti per gli utenti di SDK for Rust, mentre altri sono dettagli di implementazione:

`aws-smithy-async`

Includi questa cassa se non utilizzi Tokio per funzionalità asincrone.

`aws-smithy-runtime`

Include gli elementi costitutivi richiesti da tutti. AWS SDKs

`aws-smithy-runtime-api`

Interfacce sottostanti utilizzate dall'SDK.

`aws-smithy-types`

Tipi riesportati da altri. AWS SDKs Usalo se ne usi più di uno. SDKs

`aws-smithy-types-convert`

Funzioni di utilità per entrare e uscire `aws-smithy-types`.

Altre casse

Esistono le seguenti casse, ma non dovresti aver bisogno di sapere nulla al riguardo:

Casse relative al server di cui non hanno bisogno gli utenti di SDK for Rust:

- `aws-smithy-http-server`
- `aws-smithy-http-server-python`

Casse che contengono under-the-hood codice che gli utenti dell'SDK non devono utilizzare:

- `aws-smithy-checksum-callbacks`
- `aws-smithy-eventstream`
- `aws-smithy-http`
- `aws-smithy-protocol-test`
- `aws-smithy-query`
- `aws-smithy-json`
- `aws-smithy-xml`

Casse che non sono supportate e che verranno eliminate in futuro:

- `aws-smithy-client`
- `aws-smithy-http-auth`
- `aws-smithy-http-tower`

Cronologia dei documenti

Questo argomento descrive le modifiche importanti apportate alla AWS SDK for Rust Developer Guide nel corso della sua storia.

Modifica	Descrizione	Data
Riorganizzazione del sommario	Riorganizzazione del sommario per differenziare meglio la configurazione dall'utilizzo.	1° luglio 2024
Disponibilità generale di AWS SDK for Rust	La guida è stata aggiornata per includere nuove informazioni sulla sicurezza, esempi di codice nuovi e aggiornati, nuovi dettagli sui test unitari con esempi e altri contenuti nuovi e aggiornati per la nuova versione General Availability dell'SDK.	27 novembre 2023
Applicazione di una versione TLS minima	Sono state aggiunte informazioni su come applicare una versione di TLS nell'SDK.	4 maggio 2022
AWS SDK for Rust versione di anteprima per sviluppatori	Versione di anteprima per sviluppatori	2 dicembre 2021

Le traduzioni sono generate tramite traduzione automatica. In caso di conflitto tra il contenuto di una traduzione e la versione originale in Inglese, quest'ultima prevarrà.