

Panduan Developer

# AWS SDK for Rust



# AWS SDK for Rust: Panduan Developer

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

Merek dagang dan tampilan dagang Amazon tidak boleh digunakan sehubungan dengan produk atau layanan apa pun yang bukan milik Amazon, dengan cara apa pun yang dapat menyebabkan kebingungan antara para pelanggan, atau dengan cara apa pun yang menghina atau mendiskreditkan Amazon. Semua merek dagang lain yang tidak dimiliki oleh Amazon merupakan hak milik masing-masing pemiliknya, yang mungkin atau tidak terafiliasi, terkait dengan, atau disponsori oleh Amazon.

---

# Table of Contents

Apa itu AWS SDK for Rust? .....	1
Memulai dengan SDK .....	1
Pemeliharaan dan dukungan untuk versi SDK utama .....	1
Sumber daya tambahan .....	1
Memulai .....	3
SDKotentikasi dengan AWS .....	3
Mulai sesi portal AWS akses .....	4
Informasi otentikasi lebih lanjut .....	5
Halo tutorial .....	6
Prasyarat .....	6
Buat SDK aplikasi pertama Anda .....	6
Dasar-dasar .....	8
Prasyarat .....	6
Dasar-dasar karat .....	8
AWS SDK for Rust dasar-dasar peti .....	10
Konfigurasi proyek untuk bekerja dengan Layanan AWS .....	10
Runtime Tokio .....	11
Konfigurasi .....	12
Buat klien layanan .....	12
Konfigurasikan klien dari lingkungan .....	13
Gunakan pola pembangun untuk pengaturan khusus layanan .....	14
Konfigurasi klien eksplisit tingkat lanjut .....	15
Versi perilaku .....	16
Setel versi perilaku di <code>Cargo.toml</code> .....	16
Mengatur versi perilaku dalam kode .....	17
Penyedia kredensi .....	17
Rantai penyedia kredensi .....	17
Penyedia kredensi eksplisit .....	20
Caching identitas .....	20
Wilayah AWS seleksi .....	20
Wilayah AWS rantai penyedia .....	21
Opsi konfigurasi .....	21
Titik akhir klien .....	22
Konfigurasi Kustom .....	23

Contoh .....	26
HTTP .....	27
Hiper 1.x .....	28
Pencegat .....	30
Pendaftaran pencegat .....	31
Observabilitas .....	32
Pencatatan log .....	32
Ganti konfigurasi operasi .....	36
Percobaan ulang .....	37
Konfigurasi coba lagi default .....	38
Upaya Maksimum .....	38
Penundaan dan backoff .....	39
Mode Coba Ulang Adaptif .....	39
Timeout .....	40
APIbatas waktu .....	40
Perlindungan aliran terhenti .....	42
Gunakan SDK .....	44
Membuat permintaan .....	44
Praktik terbaik .....	45
Gunakan kembali SDK klien jika memungkinkan .....	45
Konfigurasikan API batas waktu .....	46
Konkurensi .....	46
Ketentuan .....	46
Contoh sederhana .....	47
Kepemilikan dan mutabilitas .....	48
Lebih banyak istilah! .....	49
Menulis ulang contoh kita menjadi lebih efisien (konkurensi ulir tunggal) .....	50
Menulis ulang contoh kita menjadi lebih efisien (konkurensi multi-utas) .....	52
Mendebug aplikasi multi-utas .....	54
Penanganan kesalahan .....	54
Kesalahan layanan .....	54
Metadata kesalahan .....	55
Pencetakan kesalahan terperinci dengan <code>DisplayErrorContext</code> .....	56
Buat fungsi Lambda .....	58
Paginasi .....	58
Buat presigned URLs .....	60

Dasar-dasar pra-penandatanganan .....	60
Presigning POST dan permintaan PUT .....	61
Penandatanganan Mandiri .....	62
Pengujian unit .....	63
Secara otomatis menghasilkan tiruan menggunakan mockall .....	64
Putar ulang statis .....	69
Pelayan .....	73
Contoh kode .....	75
APIGerbang .....	76
Tindakan .....	77
Skenario .....	78
AWS kontribusi komunitas .....	79
APIManajemen Gateway API .....	79
Tindakan .....	77
Penskalaan Otomatis Aplikasi .....	81
Tindakan .....	77
Aurora .....	82
Hal-hal mendasar .....	84
Tindakan .....	77
Auto Scaling .....	230
Hal-hal mendasar .....	84
Tindakan .....	77
Runtime Amazon Bedrock .....	264
Antropik Claude .....	265
Penyedia Identitas Amazon Cognito .....	280
Tindakan .....	77
Amazon Cognito Sync .....	281
Tindakan .....	77
Firehose .....	283
Tindakan .....	77
Amazon DocumentDB .....	284
Contoh nirserver .....	285
DynamoDB .....	286
Tindakan .....	77
Skenario .....	78
Contoh nirserver .....	285

AWS kontribusi komunitas .....	79
Amazon EBS .....	305
Tindakan .....	77
Amazon EC2 .....	307
Hal-hal mendasar .....	84
Tindakan .....	77
Amazon ECR .....	369
Tindakan .....	77
Amazon ECS .....	371
Tindakan .....	77
Amazon EKS .....	374
Tindakan .....	77
AWS Glue .....	376
Hal-hal mendasar .....	84
Tindakan .....	77
IAM .....	391
Hal-hal mendasar .....	84
Tindakan .....	77
AWS IoT .....	418
Tindakan .....	77
Kinesis .....	420
Tindakan .....	77
Contoh nirserver .....	285
AWS KMS .....	428
Tindakan .....	77
Lambda .....	437
Hal-hal mendasar .....	84
Tindakan .....	77
Skenario .....	78
Contoh nirserver .....	285
AWS kontribusi komunitas .....	79
MediaLive .....	486
Tindakan .....	77
MediaPackage .....	487
Tindakan .....	77
Amazon Polly .....	489

---

Tindakan .....	77
Skenario .....	78
QLDB .....	494
Tindakan .....	77
Amazon RDS .....	496
Contoh nirserver .....	285
Layanan RDS Data Amazon .....	500
Tindakan .....	77
Amazon Rekognition .....	501
Skenario .....	78
Route 53 .....	503
Tindakan .....	77
Amazon S3 .....	505
Hal-hal mendasar .....	84
Tindakan .....	77
Skenario .....	78
Contoh nirserver .....	285
SageMaker AI .....	555
Tindakan .....	77
Secrets Manager .....	558
Tindakan .....	77
Amazon SES API v2 .....	559
Tindakan .....	77
Skenario .....	78
Amazon SNS .....	575
Tindakan .....	77
Skenario .....	78
Contoh nirserver .....	285
Amazon SQS .....	581
Tindakan .....	77
Contoh nirserver .....	285
AWS STS .....	586
Tindakan .....	77
Systems Manager .....	588
Tindakan .....	77
Amazon Transcribe .....	591

---

Skenario .....	78
Keamanan .....	593
Perlindungan data .....	593
Validasi Kepatuhan .....	595
Keamanan Infrastruktur .....	596
Menegakkan versi minimum TLS .....	597
Peti yang digunakan oleh SDK .....	599
Peti pandai besi .....	599
Peti digunakan dengan SDK .....	599
Peti lainnya .....	600
Riwayat dokumen .....	601
.....	dcii



# Apa itu AWS SDK for Rust?

Rust adalah bahasa pemrograman sistem tanpa pengumpul sampah yang berfokus pada tiga tujuan: keselamatan, kecepatan, dan konkurensi.

The AWS SDK for Rust (theSDK) menyediakan Rust APIs untuk berinteraksi dengan layanan infrastruktur Amazon Web Services. Dengan menggunakan SDK, Anda dapat membangun aplikasi di atas Amazon S3, Amazon, EC2 DynamoDB, dan banyak lagi.

Topik

- [Memulai dengan SDK](#)
- [Pemeliharaan dan dukungan untuk versi SDK utama](#)
- [Sumber daya tambahan](#)

## Memulai dengan SDK

Jika Anda adalah pengguna pertama kali SDK, kami sarankan Anda mulai dengan membaca [Memulai dengan AWS SDK for Rust](#).

Untuk konfigurasi dan penyiapan, termasuk cara membuat dan mengonfigurasi klien layanan untuk membuat permintaan Layanan AWS, lihat [Konfigurasi AWS SDK for Rust](#).

Untuk informasi tentang menggunakan SDK, lihat [Gunakan AWS SDK for Rust](#).

Untuk daftar lengkap contoh kode Rust, lihat [Contoh kode](#).

## Pemeliharaan dan dukungan untuk versi SDK utama

Untuk informasi tentang pemeliharaan dan dukungan untuk versi SDK utama dan dependensi yang mendasarinya, lihat berikut ini di Panduan [Referensi Alat AWS SDKs dan Alat](#) berikut:

- [AWS SDK dan Kebijakan Pemeliharaan Alat](#)
- [AWS SDK dan Tools Version Support Matrix](#)

## Sumber daya tambahan

Selain panduan ini, berikut ini adalah sumber daya online yang berharga bagi SDK pengembang:

- [AWS SDK dan Panduan Referensi Alat](#): Berisi pengaturan, fitur, dan konsep dasar lainnya yang umum di antara AWS SDKs.
- [Situs web Rust Programming Language](#)
- [AWS SDK for Rust API Referensi](#)
- [AWS Blog Alat Pengembang](#)
- [AWS SDK for Rust kode sumber](#) pada GitHub
- [Katalog Contoh AWS Kode](#)

# Memulai dengan AWS SDK for Rust

Pelajari cara menginstal, menyiapkan, dan menggunakan aplikasi SDK untuk membuat aplikasi Rust untuk mengakses AWS sumber daya secara terprogram.

Topik

- [SDKotentikasi dengan AWS](#)
- [Halo tutorial untuk AWS SDK for Rust](#)
- [Dasar-dasar](#)

## SDKotentikasi dengan AWS

Anda harus menetapkan bagaimana kode Anda mengautentikasi AWS saat mengembangkan dengan Layanan AWS. Anda dapat mengonfigurasi akses terprogram ke AWS sumber daya dengan cara yang berbeda tergantung pada lingkungan dan AWS akses yang tersedia untuk Anda.

Untuk memilih metode otentikasi dan mengonfigurasinya SDK, lihat [Otentikasi dan akses](#) di Panduan Referensi Alat AWS SDKs dan Alat.

Kami menyarankan agar pengguna baru yang berkembang secara lokal dan tidak diberi metode otentikasi oleh majikan mereka harus disiapkan. AWS IAM Identity Center Metode ini termasuk menginstal AWS CLI untuk kemudahan konfigurasi dan untuk masuk secara teratur ke portal AWS akses. Jika Anda memilih metode ini, lingkungan Anda harus berisi elemen-elemen berikut setelah Anda menyelesaikan prosedur untuk [otentikasi Pusat IAM Identitas](#) di AWS SDKs dan Panduan Referensi Alat:

- Itu AWS CLI, yang Anda gunakan untuk memulai sesi portal AWS akses sebelum Anda menjalankan aplikasi Anda.
- [AWSconfigFile bersama](#) yang memiliki [default] profil dengan serangkaian nilai konfigurasi yang dapat direferensikan dari file. SDK Untuk menemukan lokasi file ini, lihat [Lokasi file bersama di](#) Panduan Referensi Alat AWS SDKs dan.
- `configFile bersama` menetapkan [region](#) pengaturan. Ini menetapkan default Wilayah AWS yang SDK digunakan untuk AWS permintaan. Wilayah ini digunakan untuk permintaan SDK layanan yang tidak ditentukan dengan Wilayah untuk digunakan.
- SDK Menggunakan [konfigurasi penyedia SSO token](#) profil untuk memperoleh kredensial sebelum mengirim permintaan ke. `AWSsso_role_name` Nilai, yang merupakan IAM peran yang terhubung

ke set izin Pusat IAM Identitas, memungkinkan akses ke yang Layanan AWS digunakan dalam aplikasi Anda.

configFile contoh berikut menunjukkan profil default yang diatur dengan konfigurasi penyedia SSO token. sso\_sessionPengaturan profil mengacu pada [sso-sessionbagian](#) bernama. sso-sessionBagian ini berisi pengaturan untuk memulai sesi portal AWS akses.

```
[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
```

SDKUntuk Rust tidak memerlukan paket tambahan (seperti SSO danSSO0IDC) untuk ditambahkan ke aplikasi Anda untuk menggunakan otentikasi Pusat IAM Identitas.

## Mulai sesi portal AWS akses

Sebelum menjalankan aplikasi yang mengakses Layanan AWS, Anda memerlukan sesi portal AWS akses aktif agar dapat menggunakan otentikasi Pusat IAM Identitas SDK untuk menyelesaikan kredensial. Bergantung pada panjang sesi yang dikonfigurasi, akses Anda pada akhirnya akan kedaluwarsa dan SDK akan mengalami kesalahan otentikasi. Untuk masuk ke portal AWS akses, jalankan perintah berikut di AWS CLI.

```
$ aws sso login
```

Jika Anda mengikuti panduan dan memiliki pengaturan profil default, Anda tidak perlu memanggil perintah dengan `--profile` opsi. Jika konfigurasi penyedia SSO token Anda menggunakan profil bernama, perintahnya adalah `aws sso login --profile named-profile`.

Untuk menguji secara opsional apakah Anda sudah memiliki sesi aktif, jalankan AWS CLI perintah berikut.

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

Jika sesi Anda aktif, respons terhadap perintah ini melaporkan akun Pusat IAM Identitas dan set izin yang dikonfigurasi dalam `config` file bersama.

#### Note

Jika Anda sudah memiliki sesi portal AWS akses aktif dan menjalankan `aws sso login`, Anda tidak akan diminta untuk memberikan kredensial.

Proses masuk mungkin meminta Anda untuk mengizinkan AWS CLI akses ke data Anda. Karena AWS CLI dibangun di atas SDK untuk Python, pesan izin mungkin berisi variasi `namabotocore`.

## Informasi otentikasi lebih lanjut

Pengguna manusia, juga dikenal sebagai identitas manusia, adalah orang, administrator, pengembang, operator, dan konsumen aplikasi Anda. Mereka harus memiliki identitas untuk mengakses AWS lingkungan dan aplikasi Anda. Pengguna manusia yang merupakan anggota organisasi Anda - itu berarti Anda, pengembang - dikenal sebagai identitas tenaga kerja.

Gunakan kredensi sementara saat mengakses. AWS Anda dapat menggunakan penyedia identitas bagi pengguna manusia Anda untuk menyediakan akses gabungan ke AWS akun dengan mengambil peran, yang menyediakan kredensi sementara. Untuk manajemen akses terpusat, kami menyarankan Anda menggunakan AWS IAM Identity Center (Pusat IAM Identitas) untuk mengelola akses ke akun Anda dan izin dalam akun tersebut. Untuk alternatif lainnya, lihat yang berikut ini:

- Untuk mempelajari lebih lanjut tentang praktik terbaik, lihat [Praktik terbaik keamanan IAM di Panduan IAM Pengguna](#).
- Untuk membuat AWS kredensial jangka pendek, lihat [Kredensial Keamanan Sementara di Panduan Pengguna. IAM](#)
- Untuk mempelajari tentang penyedia kredensi lain yang didukung oleh SDK for Rust, lihat Penyedia [kredensi terstandarisasi di Panduan Referensi](#) Alat AWS SDK dan Alat.

# Halo tutorial untuk AWS SDK for Rust

## Prasyarat

Untuk menggunakan AWS SDK for Rust, Anda harus menginstal Rust and Cargo.

- Instal rantai alat Rust: <https://www.rust-lang.org/tools/install>
- Instal cargo-component [alat](#) dengan menjalankan perintah: `cargo install cargo-component`

## Alat yang direkomendasikan:

Alat opsional berikut dapat diinstal di Anda IDE untuk membantu penyelesaian kode dan pemecahan masalah.

- Ekstensi rust-analyzer, lihat [Rust di Visual Studio Code](#).
- Pengembang Amazon Q, lihat [Menginstal ekstensi atau plugin Pengembang Amazon Q di situs Anda IDE](#).

## Buat SDK aplikasi pertama Anda

Prosedur ini membuat aplikasi Rust pertama SDK Anda yang mencantumkan tabel DynamoDB Anda.

1. Di jendela terminal atau konsol, navigasikan ke lokasi di komputer tempat Anda ingin membuat aplikasi.
2. Jalankan perintah berikut untuk membuat `hello_world` direktori dan mengisinya dengan proyek kerangka Rust:

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

3. Arahkan ke `hello_world` direktori dan gunakan perintah berikut untuk menambahkan dependensi yang diperlukan ke aplikasi:

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

Dependensi ini termasuk SDK peti yang menyediakan fitur konfigurasi dan dukungan untuk DynamoDB, termasuk peti, yang digunakan untuk [tokioimplementasikan operasi I/O asinkron](#).

#### Note

Kecuali jika Anda menggunakan fitur seperti tokio/full Tokio tidak akan menyediakan runtime async. The SDK for Rust membutuhkan runtime async.

#### 4. Perbarui main.rs di src direktori untuk berisi kode berikut.

```
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

Contoh ini hanya menampilkan halaman pertama hasil. Lihat [the section called “Paginasi”](#) untuk mempelajari cara menangani beberapa halaman hasil.

## 5. Jalankan program:

```
$ cargo run
```

Anda akan melihat daftar nama tabel Anda.

## Dasar-dasar

### Prasyarat

Untuk menggunakan AWS SDK for Rust, Anda harus menginstal Rust and Cargo.

- Instal rantai alat Rust: <https://www.rust-lang.org/tools/install>
- Instal cargo-component [alat](#) dengan menjalankan perintah: `cargo install cargo-component`

### Alat yang direkomendasikan:

Alat opsional berikut dapat diinstal di Anda IDE untuk membantu penyelesaian kode dan pemecahan masalah.

- Ekstensi rust-analyzer, lihat [Rust di Visual Studio Code](#).
- Pengembang Amazon Q, lihat [Menginstal ekstensi atau plugin Pengembang Amazon Q di IDE](#).

### Dasar-dasar karat

Berikut ini adalah beberapa dasar-dasar bahasa pemrograman Rust yang akan membantu untuk diketahui. Semua referensi untuk informasi lebih lanjut berasal dari [The Rust Programming Language](#).



- `Cargo.toml` adalah file konfigurasi proyek Rust standar, berisi dependensi dan beberapa metadata tentang proyek. File sumber Rust memiliki ekstensi `.rs` file. Lihat [Halo, Cargo!](#).
- `Cargo.toml` dapat disesuaikan dengan profil, lihat [Menyesuaikan Build dengan Profil Rilis](#). Profil ini sama sekali tidak terkait dan independen dari AWS penggunaan profil dalam AWS config file bersama.
- Cara umum untuk menambahkan dependensi perpustakaan ke proyek Anda dan file ini adalah dengan menggunakan `cargo add` Lihat [cargo-add](#).
- `let` memiliki struktur fungsi dasar seperti berikut ini. `let` kata kunci mendeklarasikan variabel dan mungkin dipasangkan dengan penugasan (`=`). Jika Anda tidak menentukan tipe setelahnya `let`, maka kompiler akan menyimpulkan satu. Lihat [Variabel dan Mutabilitas](#).

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

- Untuk mendeklarasikan variabel `x` dengan tipe eksplisit `T`, Rust menggunakan sintaks `x: T` Lihat [Tipe Data](#).
- `struct X {}` mendefinisikan tipe `X` baru. Metode diimplementasikan pada tipe `X` struct khusus. Metode untuk tipe `X` dideklarasikan dengan blok implementasi diawali dengan kata kunci `impl`. Di dalam blok implementasi, `self` mengacu pada instance struct tempat metode dipanggil. Lihat [Kata Kunci impl](#) dan [Sintaks Metode](#).
- Jika tanda seru (`!`) mengikuti apa yang tampaknya menjadi definisi fungsi atau panggilan fungsi, maka kode mendefinisikan atau memanggil makro. Lihat [Makro](#).
- Di Rust, kesalahan yang tidak dapat dipulihkan diwakili oleh makro `panic!` Ketika sebuah program bertemu dengan program `panic!` itu akan berhenti berjalan, mencetak pesan kegagalan, melepas leleh, membersihkan tumpukan, dan berhenti. Lihat Kesalahan [yang Tidak Dapat Dipulihkan](#) dengan `panic!`
- Rust tidak mendukung pewarisan fungsionalitas dari kelas dasar seperti bahasa pemrograman lainnya; `traits` adalah bagaimana Rust menyediakan metode yang berlebihan. Sifat mungkin dianggap secara konseptual mirip dengan antarmuka. Namun, sifat dan antarmuka sejati memiliki perbedaan dan sering digunakan secara berbeda dalam proses desain. Lihat [Sifat: Mendefinisikan Perilaku Bersama](#).
- Polimorfisme mengacu pada kode yang mendukung fungsionalitas untuk beberapa tipe data tanpa harus menulis masing-masing. `let` mendukung polimorfisme melalui enum, sifat, dan obat generik. Lihat [Warisan sebagai Sistem Tipe dan sebagai Berbagi Kode](#).

- Rust sangat eksplisit tentang memori. Smart pointer “adalah struktur data yang bertindak seperti pointer tetapi juga memiliki metadata dan kemampuan tambahan”. Lihat [Smart Pointer](#).
- Jenisnya Cow adalah penunjuk clone-on-write pintar yang membantu mentransfer kepemilikan memori ke penelepon bila diperlukan. Lihat [Enum std::borrow::Cow](#).
- Tipenya Arc adalah penunjuk pintar Terhitung Referensi Atom yang menghitung instance yang dialokasikan. Lihat [Struct std::sync::Arc](#).
- The SDK for Rust sering menggunakan pola pembangun untuk membangun tipe yang kompleks.

## AWS SDK for Rust dasar-dasar peti

- Pustaka SDK for Rust dipisahkan menjadi peti perpustakaan yang berbeda oleh masing-masing Layanan AWS. Peti ini tersedia di <https://docs.rs/>.
- Layanan AWS peti mengikuti konvensi penamaan `aws-sdk-[servicename]`, seperti `aws-sdk-s3` dan `aws-sdk-dynamodb`.
- Peti inti utama untuk fungsionalitas SDK for Rust adalah `aws-config`. Ini termasuk dalam sebagian besar proyek karena menyediakan fungsionalitas untuk membaca konfigurasi dari lingkungan.
  - Jangan bingung ini dengan Layanan AWS yang disebut AWS Config. Karena itu adalah layanan, ia mengikuti konvensi standar dan disebut `aws-sdk-config`.

## Konfigurasi proyek untuk bekerja dengan Layanan AWS

- Anda perlu menambahkan peti ke proyek Anda untuk setiap Layanan AWS yang Anda ingin aplikasi Anda gunakan.
- Cara yang disarankan untuk menambahkan peti adalah menggunakan baris perintah di direktori proyek Anda dengan menjalankannya `cargo add [crateName]`, seperti `cargo add aws-sdk-s3`.
  - Ini akan menambahkan baris ke proyek Anda di `Cargo.toml` bawah `[dependencies]`.
  - Secara default, ini akan menambahkan versi terbaru peti ke proyek Anda.
- Dalam file sumber Anda, gunakan `use` pernyataan untuk membawa item dari peti mereka ke dalam ruang lingkup. Lihat [Menggunakan Paket Eksternal](#) di situs web Bahasa Pemrograman Rust.
  - Nama peti sering diberi tanda hubung, tetapi tanda hubung diubah menjadi garis bawah saat benar-benar menggunakan peti. Misalnya, `aws-config` peti digunakan dalam `use` pernyataan kode sebagai `use aws_config`.

- Konfigurasi adalah topik yang kompleks. Konfigurasi dapat terjadi secara langsung dalam kode, atau ditentukan secara eksternal dalam variabel lingkungan atau file konfigurasi. Untuk informasi selengkapnya, lihat [Opsi konfigurasi](#).
- Saat SDK memuat konfigurasi Anda, nilai yang tidak valid dicatat alih-alih menghentikan eksekusi karena sebagian besar pengaturan memiliki default yang masuk akal. Untuk mempelajari cara mengaktifkan logging, lihat [Aktifkan pencatatan AWS SDK for Rust kode](#).
- Sebagian besar variabel lingkungan dan pengaturan file konfigurasi dimuat sekali ketika program Anda dimulai. Setiap pembaruan pada nilai tidak akan terlihat sampai Anda memulai ulang program Anda.

## Runtime Tokio

- Tokio adalah runtime asinkron untuk bahasa pemrograman Rust, SDK ia menjalankan tugas. async [Lihat tokio.rs dan docs.rs/tokio](#).
- The SDK for Rust membutuhkan runtime async. Kami menyarankan Anda menambahkan peti berikut ke proyek Anda:

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

- `tokio::main` atribut makro menciptakan titik masuk utama async ke program Anda. Untuk menggunakan makro ini, tambahkan ke baris sebelum `main` metode Anda, seperti yang ditunjukkan pada berikut ini:

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

# Konfigurasi AWS SDK for Rust

Pelajari cara mengkonfigurasi file AWS SDK for Rust. Anda harus menetapkan bagaimana kode Anda mengautentikasi dengan AWS ketika Anda mengembangkan dengan Layanan AWS. Anda juga harus mengatur yang ingin Wilayah AWS Anda gunakan.

[Panduan Referensi AWS SDKs and Tools](#) juga berisi pengaturan, fitur, dan konsep dasar lainnya yang umum di antara banyak. AWS SDKs

## Topik

- [Buat klien layanan](#)
- [Versi perilaku](#)
- [Penyedia Kredensi](#)
- [Wilayah AWS seleksi](#)
- [Opsi konfigurasi](#)
- [Titik akhir klien](#)
- [HTTP](#)
- [Pencegat](#)
- [Gunakan fitur observabilitas](#)
- [Ganti konfigurasi operasi tunggal klien](#)
- [Percobaan ulang](#)
- [Timeout](#)

## Buat klien layanan

Untuk membuat permintaan ke Layanan AWS, pertama-tama Anda membuat instance klien untuk layanan itu. Anda dapat mengonfigurasi pengaturan umum untuk klien layanan seperti batas waktu, HTTP klien, dan konfigurasi coba lagi.

Setiap klien layanan membutuhkan Wilayah AWS dan penyedia kredensi. SDK menggunakan nilai-nilai ini untuk mengirim permintaan ke Wilayah yang benar untuk sumber daya Anda dan untuk menandatangani permintaan dengan kredensial yang benar. Anda dapat menentukan nilai-nilai ini secara terprogram dalam kode atau membuatnya dimuat secara otomatis dari lingkungan.

**Note**

Pelanggan layanan bisa mahal untuk dibangun dan umumnya dimaksudkan untuk dibagikan. Untuk memfasilitasi ini, semua `Client` struct menerapkan `Clone`.

Ini SDK memiliki serangkaian tempat (atau sumber) yang diperiksa untuk menemukan nilai untuk pengaturan konfigurasi.

1. Pengaturan eksplisit apa pun yang disetel dalam kode atau pada klien layanan itu sendiri lebih diutamakan daripada yang lain.
2. Variabel-variabel lingkungan
  - Untuk detail tentang pengaturan variabel lingkungan, lihat [variabel lingkungan](#) di Panduan Referensi Alat AWS SDKs dan Alat.
3. Berbagi config dan credentials file
  - Untuk detail tentang menyiapkan file-file ini, lihat [Dibagikan config dan credentials file](#) di Panduan Referensi Alat AWS SDKs dan Alat.
4. Setiap nilai default yang disediakan oleh kode SDK sumber itu sendiri digunakan terakhir.
  - Beberapa properti, seperti Region, tidak memiliki default. Anda harus menentukannya secara eksplisit dalam kode, dalam pengaturan lingkungan, atau dalam file bersamaconfig. Jika tidak SDK dapat menyelesaikan konfigurasi yang diperlukan, API permintaan dapat gagal saat runtime.

Sebagian besar pengaturan variabel lingkungan dan pengaturan credentials file config dan dibagikan oleh beberapa AWS SDKs dan alat untuk perilaku yang konsisten. Untuk melihat semua pengaturan yang SDK dapat diselesaikan dari variabel lingkungan atau file konfigurasi, lihat [referensi Pengaturan di Panduan Referensi](#) Alat AWS SDKs dan.

## Konfigurasi klien dari lingkungan

Untuk membuat klien dengan konfigurasi bersumber lingkungan, gunakan metode statis dari peti: `aws-config`

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

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

Membuat klien dengan cara ini berguna saat berjalan di Amazon Elastic Compute Cloud AWS Lambda, atau konteks lain di mana konfigurasi klien layanan tersedia langsung dari lingkungan. Ini memisahkan kode Anda dari lingkungan tempat ia berjalan dan membuatnya lebih mudah untuk menerapkan aplikasi Anda ke beberapa Wilayah AWS tanpa mengubah kode.

Anda dapat secara eksplisit mengganti properti tertentu. Konfigurasi eksplisit lebih diutamakan daripada konfigurasi yang diselesaikan dari lingkungan eksekusi. Contoh berikut memuat konfigurasi dari lingkungan, tetapi secara eksplisit mengesampingkan: Wilayah AWS

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

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

#### Note

Tidak semua nilai konfigurasi bersumber oleh klien pada waktu pembuatan. Pengaturan terkait kredensial, seperti kunci akses sementara dan konfigurasi Pusat IAM Identitas, diakses oleh lapisan penyedia kredensi ketika klien digunakan untuk membuat permintaan.

Kode yang `BehaviorVersion::latest()` ditampilkan dalam contoh sebelumnya menunjukkan versi yang akan digunakan SDK untuk default. `BehaviorVersion::latest()` sesuai untuk sebagian besar kasus. Untuk detailnya, lihat [Versi perilaku](#).

## Gunakan pola pembangun untuk pengaturan khusus layanan

Ada beberapa opsi yang hanya dapat dikonfigurasi pada jenis klien layanan tertentu. Namun, paling sering, Anda masih ingin memuat sebagian besar konfigurasi dari lingkungan, dan kemudian secara khusus menambahkan opsi tambahan. Pola pembangun adalah pola umum di dalam AWS SDK for Rust peti. Pertama-tama Anda memuat konfigurasi umum menggunakan `aws_config::defaults`, kemudian menggunakan `from` metode untuk memuat konfigurasi itu ke pembuat untuk layanan yang Anda kerjakan. Anda kemudian dapat mengatur nilai konfigurasi unik apa pun untuk layanan dan panggilan tersebut `build`. Terakhir, klien dibuat dari konfigurasi termodifikasi ini.

```
// 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);
```

Salah satu cara untuk menemukan metode tambahan yang tersedia untuk jenis klien layanan tertentu adalah dengan menggunakan API dokumentasi, seperti untuk [aws\\_sdk\\_s3::config::Builder](#).

## Konfigurasi klien eksplisit tingkat lanjut

Untuk mengonfigurasi klien layanan dengan nilai tertentu alih-alih memuat konfigurasi dari lingkungan, Anda dapat menentukannya pada `Config` pembuat klien seperti yang ditunjukkan dalam berikut ini:

```
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);
```

Saat Anda membuat konfigurasi layanan dengan `aws_sdk_s3::Config::builder()`, tidak ada konfigurasi default yang dimuat. Default hanya dimuat saat membuat konfigurasi berdasarkan `aws_config::defaults`

Ada beberapa opsi yang hanya dapat dikonfigurasi pada jenis klien layanan tertentu. Contoh sebelumnya menunjukkan contoh ini dengan menggunakan `endpoint_resolver` fungsi pada klien Amazon S3.

## Versi perilaku

AWS SDK for Rust pengembang mengharapkan dan mengandalkan perilaku yang kuat dan dapat diprediksi yang ditawarkan bahasa dan perpustakaan utamanya. Untuk membantu pengembang yang menggunakan SDK for Rust mendapatkan perilaku yang diharapkan, konfigurasi klien diperlukan untuk menyertakan `BehaviorVersion` file. `BehaviorVersion` Menentukan versi yang SDK defaultnya diharapkan. Hal ini memungkinkan SDK perkembangan dari waktu ke waktu, mengubah praktik terbaik agar sesuai dengan standar baru dan mendukung fitur baru tanpa dampak buruk yang tidak terduga pada perilaku aplikasi Anda.

### Warning

Jika Anda mencoba mengkonfigurasi SDK atau membuat klien tanpa secara eksplisit menentukan `aBehaviorVersion`, konstruktor akan panic.

Misalnya, bayangkan versi baru dirilis dengan kebijakan coba ulang default yang baru. SDK Jika aplikasi Anda menggunakan `BehaviorVersion` pencocokan versi sebelumnya SDK, maka konfigurasi sebelumnya digunakan sebagai ganti konfigurasi default yang baru.

Setiap kali versi perilaku baru SDK untuk Rust dirilis, yang sebelumnya `BehaviorVersion` ditandai dengan `deprecated` atribut SDK for Rust dan versi baru ditambahkan. Hal ini menyebabkan peringatan terjadi pada waktu kompilasi, tetapi sebaliknya, biarkan build berlanjut seperti biasa. `BehaviorVersion::latest()` juga diperbarui untuk menunjukkan perilaku default versi baru.

### Note

Dalam kebanyakan kasus, Anda harus menggunakan `BehaviorVersion::latest()` kode atau tanda fitur `behavior-version-latest` dalam `Cargo.toml` file. Disarankan untuk menyematkan ke versi tertentu hanya selama diperlukan.

## Setel versi perilaku di `Cargo.toml`

Anda dapat menentukan versi perilaku untuk SDK dan modul individual, seperti `aws-sdk-s3` atau `aws-sdk-iam`, dengan menyertakan tanda fitur yang sesuai dalam `Cargo.toml` file. Saat ini, hanya `latest` versi yang SDK didukung di `Cargo.toml`:



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

## Mengatur versi perilaku dalam kode

Kode Anda dapat mengubah versi perilaku sesuai kebutuhan dengan menentukannya saat mengonfigurasi SDK atau klien:

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

Contoh ini membuat konfigurasi yang menggunakan lingkungan untuk mengkonfigurasi SDK tetapi menyetel `BehaviorVersion` ke `v2023_11_09()`.

## Penyedia Kredensi

Untuk membuat permintaan untuk AWS menggunakan AWS SDK for Rust, menggunakan kredensial yang SDK ditandatangani secara kriptografi yang dikeluarkan oleh AWS. Saat runtime, SDK mengambil nilai konfigurasi untuk kredensial dengan memeriksa beberapa lokasi.

Jika konfigurasi yang diambil menyertakan [setelan akses masuk AWS IAM Identity Center tunggal](#), maka akan SDK berfungsi dengan Pusat IAM Identitas untuk mengambil kredensial sementara yang digunakan untuk membuat permintaan. Layanan AWS

Jika konfigurasi yang diambil menyertakan [kredensial sementara](#), SDK menggunakannya untuk melakukan panggilan. Layanan AWS Kredensi sementara terdiri dari kunci akses dan token sesi.

Otentikasi dengan AWS dapat ditangani di luar basis kode Anda. Banyak metode otentikasi dapat dideteksi, digunakan, dan disegarkan secara otomatis dengan SDK menggunakan rantai penyedia kredensi.

Untuk opsi terpandu untuk memulai AWS autentikasi untuk proyek Anda, lihat [Otentikasi dan akses di Panduan Referensi Alat AWS SDKs dan Alat](#).

## Rantai penyedia kredensi

Jika Anda tidak secara eksplisit menentukan penyedia kredensi saat membuat klien, SDK for Rust menggunakan rantai penyedia kredensi yang memeriksa serangkaian tempat di mana Anda dapat

menyediakan kredensial. Setelah SDK menemukan kredensial di salah satu lokasi ini, pencarian berhenti. Untuk detail tentang membangun klien, lihat [Buat klien layanan](#).

Contoh berikut tidak menentukan penyedia kredensial dalam kode. SDK menggunakan rantai penyedia kredensial untuk mendeteksi otentikasi yang telah diatur di lingkungan hosting, dan menggunakan otentikasi itu untuk panggilan ke Layanan AWS

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

## Urutan pengambilan kredensial

Rantai penyedia kredensial mencari kredensial menggunakan urutan yang telah ditentukan berikut ini:

### 1. Akses variabel lingkungan kunci

SDK upaya untuk memuat kredensial dari variabel `AWS_ACCESS_KEY_ID` dan `AWS_SECRET_ACCESS_KEY`, dan `AWS_SESSION_TOKEN` lingkungan.

### 2. Yang dibagikan AWS **config** dan **credentials** file

SDK upaya untuk memuat kredensial dari [default] profil di bersama AWS config dan credentials file. Anda dapat menggunakan variabel `AWS_PROFILE` lingkungan untuk memilih profil bernama yang SDK ingin Anda muat alih-alih menggunakan [default]. credentialsFile config dan dibagikan oleh berbagai AWS SDKs alat. Untuk informasi selengkapnya tentang file-file ini, lihat [Dibagikan config dan credentials file](#) di Panduan Referensi Alat AWS SDKs dan Alat.

Jika Anda menggunakan Pusat IAM Identitas untuk mengautentikasi, ini adalah saat SDK untuk Rust menggunakan token masuk tunggal yang disiapkan dengan menjalankan perintah. `aws sso login` SDK menggunakan kredensial sementara yang ditukar oleh Pusat IAM Identitas dengan token yang valid. SDK kemudian menggunakan kredensial sementara saat memanggil Layanan AWS. Untuk informasi terperinci tentang proses ini, lihat [Memahami resolusi SDK kredensial untuk Layanan AWS](#) di Panduan Referensi Alat AWS SDKs dan Alat.

- Untuk panduan tentang mengonfigurasi penyedia ini, lihat [otentikasi Pusat IAM Identitas](#) di Panduan Referensi Alat AWS SDKs dan Alat.
- Untuk detail tentang properti SDK konfigurasi untuk penyedia ini, lihat Penyedia [kredensial Pusat IAM Identitas](#) di Panduan Referensi Alat AWS SDKs dan Alat.

### 3. AWS STS identitas web

Saat membuat aplikasi seluler atau aplikasi web berbasis klien yang memerlukan akses ke AWS, AWS Security Token Service (AWS STS) mengembalikan satu set kredensi keamanan sementara untuk pengguna federasi yang diautentikasi melalui penyedia identitas publik (iDP).

- Ketika Anda menentukan ini dalam profil, SDK atau alat mencoba untuk mengambil kredensi sementara menggunakan metode. AWS STS `AssumeRoleWithWebIdentity` API Untuk detail tentang metode ini, lihat [AssumeRoleWithWebIdentity](#) di AWS Security Token Service API Referensi.
- Untuk panduan tentang mengonfigurasi penyedia ini, lihat Menggabungkan [dengan identitas web atau OpenID Connect](#) di Panduan Referensi Alat AWS SDK dan OpenID.
- Untuk detail tentang properti SDK konfigurasi untuk penyedia ini, lihat [Mengasumsikan penyedia kredensi peran](#) di Panduan Referensi Alat AWS SDK dan.

#### 4. Kredensi EKS wadah Amazon ECS dan Amazon

Tugas Amazon Elastic Container Service dan akun layanan Kubernetes Anda dapat memiliki IAM peran yang terkait dengannya. Izin yang diberikan dalam IAM peran diasumsikan oleh kontainer yang berjalan di tugas atau kontainer pod. Peran ini memungkinkan Anda SDK untuk kode aplikasi Rust (pada wadah) untuk menggunakan yang lain Layanan AWS.

SDK Upaya untuk mengambil kredensial dari `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` atau variabel `AWS_CONTAINER_CREDENTIALS_FULL_URI` lingkungan, yang dapat diatur secara otomatis oleh Amazon dan ECS Amazon. EKS

- Untuk detail tentang menyiapkan peran ini untuk Amazon ECS, lihat Peran [ECStugas IAM Amazon](#) di Panduan Pengembang Layanan Kontainer Elastis Amazon.
- Untuk informasi EKS penyiapan Amazon, lihat [Menyiapkan Agen Identitas EKS Pod Amazon](#) di Panduan EKS Pengguna Amazon.
- Untuk detail tentang properti SDK konfigurasi untuk penyedia ini, lihat Penyedia [kredensi kontainer](#) di Panduan Referensi Alat AWS SDKs dan Alat.

#### 5. Layanan EC2 Metadata Instans Amazon

Buat IAM peran dan lampirkan ke instance Anda. Aplikasi SDK for Rust pada instance mencoba mengambil kredensial yang disediakan oleh peran dari metadata instance.

- The SDK for Rust hanya mendukung [IMDSv2](#).
- Untuk detail tentang pengaturan peran ini dan menggunakan metadata, [IAMperan untuk Amazon EC2](#) dan [Bekerja dengan metadata instans di Panduan Pengguna](#) Amazon. EC2

- Untuk detail tentang properti SDK konfigurasi untuk penyedia ini, lihat penyedia [IMDSkredensi](#) di Panduan Referensi Alat AWS SDKs dan Alat.

6. Jika kredensil masih belum diselesaikan pada saat ini, operasi panics dengan kesalahan.

Untuk detail tentang setelan konfigurasi penyedia AWS kredensi, lihat [Penyedia kredensi terstandarisasi](#) di referensi Pengaturan AWS SDKs dan Panduan Referensi Alat.

## Penyedia kredensi eksplisit

Alih-alih mengandalkan rantai penyedia kredensi untuk mendeteksi metode otentikasi Anda, Anda dapat menentukan penyedia kredensi tertentu yang harus digunakan. SDK Saat memuat konfigurasi umum menggunakan `aws_config::defaults`, Anda dapat menentukan penyedia kredensi khusus seperti yang ditunjukkan pada berikut ini:

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

Anda dapat menerapkan penyedia kredensi Anda sendiri dengan menerapkan [ProvideCredentials](#) sifat tersebut.

## Caching identitas

SDK Will cache kredensil dan jenis identitas lainnya seperti SSO token. Secara default, aplikasi SDK menggunakan implementasi cache malas yang memuat kredensil pada permintaan pertama, menyimpannya di cache, dan kemudian mencoba untuk menyegarkannya selama permintaan lain saat hampir kedaluwarsa. Klien yang dibuat dari yang sama `SdkConfig` akan berbagi [IdentityCache](#).

## Wilayah AWS seleksi

Anda dapat mengakses Layanan AWS yang beroperasi di wilayah geografis tertentu dengan menggunakan Wilayah AWS. Ini dapat berguna baik untuk redundansi dan untuk menjaga data dan aplikasi Anda berjalan dekat dengan tempat Anda dan pengguna Anda mengaksesnya. Untuk informasi selengkapnya tentang cara Wilayah digunakan, lihat [Wilayah AWS](#) di Panduan Referensi Alat AWS SDKs dan Alat.

### ⚠ Important

Sebagian besar sumber daya berada di tempat tertentu Wilayah AWS dan Anda harus menyediakan Wilayah yang tepat untuk sumber daya saat menggunakan SDK

## Wilayah AWS rantai penyedia

Proses pencarian berikut digunakan saat memuat konfigurasi klien layanan dari lingkungan eksekusi. Nilai pertama yang SDK ditemukan set digunakan dalam konfigurasi klien. Untuk informasi selengkapnya tentang membuat klien layanan, lihat [Konfigurasi klien dari lingkungan](#).

1. Setiap Wilayah eksplisit diatur secara terprogram.
2. Variabel `AWS_REGION` lingkungan diperiksa.
  - Jika Anda menggunakan AWS Lambda layanan, variabel lingkungan ini diatur secara otomatis oleh AWS Lambda wadah.
3. `regionProperti` dalam AWS config file bersama dicentang.
  - Variabel `AWS_CONFIG_FILE` lingkungan dapat digunakan untuk mengubah lokasi config file bersama. Untuk mempelajari lebih lanjut tentang tempat penyimpanan file ini, lihat [Lokasi credentials file bersama config dan](#) di Panduan Referensi Alat AWS SDKs dan.
  - Variabel `AWS_PROFILE` lingkungan dapat digunakan untuk memilih profil bernama bukan default. Untuk mempelajari lebih lanjut tentang mengonfigurasi profil yang berbeda, lihat [Dibagikan config dan credentials file](#) di Panduan Referensi Alat AWS SDKs dan Alat.
4. SDKUpaya menggunakan Layanan Metadata EC2 Instans Amazon untuk menentukan Wilayah instans Amazon EC2 yang sedang berjalan.
  - AWS SDK for Rust Satu-satunya dukunganIMDSv2.

## Opsi konfigurasi

Banyak pengaturan konfigurasi dapat ditangani di luar kode Anda. Sebagian besar pengaturan konfigurasi dapat diatur sebagai variabel lingkungan atau dalam AWS config file bersama yang terpisah. `configFile` bersama dapat mempertahankan set pengaturan terpisah, yang disebut profil, untuk menyediakan konfigurasi yang berbeda untuk lingkungan atau pengujian yang berbeda.

Variabel lingkungan dan pengaturan config file bersama distandarisasi dan dibagikan di seluruh AWS SDKs dan alat untuk mendukung fungsionalitas yang konsisten di berbagai bahasa.

Lihat Panduan Referensi Alat AWS SDKs dan untuk mempelajari tentang mengonfigurasi aplikasi Anda melalui metode ini, ditambah detail pada setiap setelan lintas sdk. Beberapa halaman yang menarik mungkin:

- [Berbagi config dan credentials file](#) - Menjelaskan profil dan format file konfigurasi ini.
- [Lokasi yang dibagikan config dan credentials file](#) - Menjelaskan lokasi default file-file ini dan cara mengubahnya.
- [Dukungan variabel lingkungan](#) - Menjelaskan pengaturan variabel lingkungan.
- [Pengaturan referensi](#) - Informasi referensi tentang semua SDK pengaturan silang.

## Titik akhir klien

Saat AWS SDK for Rust memanggil Layanan AWS, salah satu langkah pertamanya adalah menentukan ke mana harus merutekan permintaan. Proses ini dikenal sebagai resolusi titik akhir.

Anda dapat mengonfigurasi resolusi titik akhir untuk SDK saat Anda membuat klien layanan. Konfigurasi default untuk resolusi titik akhir biasanya baik-baik saja, tetapi ada beberapa alasan mengapa Anda mungkin ingin mengubah konfigurasi default. Dua contoh alasannya adalah sebagai berikut:

- Untuk membuat permintaan ke versi pra-rilis layanan atau untuk penyebaran lokal layanan.
- Untuk mengakses ke fitur layanan tertentu yang belum dimodelkan di SDK

### Warning

Resolusi titik akhir adalah SDK topik lanjutan. Jika Anda mengubah pengaturan default, Anda berisiko melanggar kode Anda. Pengaturan default berlaku untuk sebagian besar pengguna di lingkungan produksi.

Titik akhir kustom dapat diatur secara global sehingga digunakan untuk semua permintaan layanan, atau Anda dapat menetapkan titik akhir khusus untuk yang spesifik. Layanan AWS

Endpoint kustom dapat dikonfigurasi menggunakan variabel lingkungan atau pengaturan dalam AWS config file bersama. Untuk informasi tentang pendekatan ini, lihat [Titik akhir khusus layanan](#) di Panduan Referensi Alat AWS SDKs dan Alat. Untuk daftar lengkap setelan config file bersama

dan variabel lingkungan untuk semua Layanan AWS, lihat [Pengidentifikasi untuk titik akhir khusus layanan](#).

Atau, kustomisasi ini juga dapat dikonfigurasi dalam kode Anda, seperti yang ditunjukkan pada bagian berikut.

## Konfigurasi Kustom

Anda dapat menyesuaikan resolusi titik akhir klien layanan dengan dua metode yang tersedia saat Anda membangun klien:

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

Anda dapat mengatur kedua properti. Namun, paling sering Anda hanya menyediakan satu. Untuk penggunaan umum, paling `endpoint_url` sering disesuaikan.

### Tetapkan titik akhir URL

Anda dapat menetapkan nilai untuk `endpoint_url` untuk menunjukkan nama host “dasar” untuk layanan. Nilai ini, bagaimanapun, tidak final karena diteruskan sebagai parameter ke `ResolveEndpoint` instance klien. `ResolveEndpointImplementasi` kemudian dapat memeriksa dan berpotensi memodifikasi nilai tersebut untuk menentukan titik akhir.

### Tetapkan penyelesai titik akhir

`ResolveEndpointImplementasi` klien layanan menentukan titik akhir yang diselesaikan yang SDK digunakan untuk setiap permintaan yang diberikan. Klien layanan memanggil `resolve_endpoint` metode untuk setiap permintaan dan menggunakan [EndpointFuture](#) nilai yang dikembalikan oleh resolver tanpa perubahan lebih lanjut.

Contoh berikut menunjukkan penyediaan implementasi penyelesai titik akhir kustom untuk klien Amazon S3 yang menyelesaikan titik akhir per tahap yang berbeda, seperti pementasan dan produksi:

```
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

Resolver titik akhir, dan dengan ekstensi `ResolveEndpoint` sifat, khusus untuk setiap layanan, dan dengan demikian hanya dapat dikonfigurasi pada konfigurasi klien layanan. `EndpointURL`, di sisi lain, dapat dikonfigurasi baik menggunakan konfigurasi bersama (berlaku untuk semua layanan yang berasal darinya) atau untuk layanan tertentu.

## ResolveEndpoint parameter

`resolve_endpoint` Metode ini menerima parameter khusus layanan yang berisi properti yang digunakan dalam resolusi titik akhir.

Setiap layanan mencakup properti dasar berikut:

Nama	Tipe	Deskripsi
<code>region</code>	String	Klien Wilayah AWS



Nama	Tipe	Deskripsi
endpoint	String	Sebuah representasi string dari nilai set <code>endpointUrl</code>
use_fips	Boolean	Apakah FIPS titik akhir diaktifkan dalam konfigurasi klien
use_dual_stack	Boolean	Apakah titik akhir dual-stack diaktifkan dalam konfigurasi klien

Layanan AWS dapat menentukan properti tambahan yang diperlukan untuk resolusi. Misalnya, [parameter endpoint](#) Amazon S3 menyertakan nama bucket dan juga beberapa pengaturan fitur khusus Amazon S3. Misalnya, `force_path_style` properti menentukan apakah pengalamatan host virtual dapat digunakan.

Jika Anda menerapkan penyedia Anda sendiri, Anda tidak perlu membuat instance parameter endpoint Anda sendiri. SDKIni menyediakan properti untuk setiap permintaan dan meneruskannya ke implementasi `Andaresolve_endpoint`.

## Bandingkan menggunakan `endpoint_url` dengan menggunakan `endpoint_resolver`

Penting untuk dipahami bahwa dua konfigurasi berikut, satu yang menggunakan `endpoint_url` dan yang lain yang menggunakan `endpoint_resolver`, NOT menghasilkan klien dengan perilaku resolusi titik akhir yang setara.

```
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();
```

Klien yang menetapkan `endpoint_url` menentukan basis URL yang diteruskan ke penyedia (default), yang dapat dimodifikasi sebagai bagian dari resolusi titik akhir.

Klien yang menetapkan `endpoint_resolver` menentukan final yang digunakan URL klien Amazon S3.

## Contoh

Endpoint kustom sering digunakan untuk pengujian. Alih-alih melakukan panggilan ke layanan berbasis cloud, panggilan dialihkan ke layanan simulasi yang dihosting secara lokal. Dua opsi tersebut adalah:

- [DynamoDB](#) lokal — versi lokal dari layanan Amazon DynamoDB.
- [LocalStack](#)— emulator layanan cloud yang berjalan dalam wadah di mesin lokal Anda.

Contoh berikut mengilustrasikan dua cara berbeda untuk menentukan titik akhir kustom untuk menggunakan dua opsi pengujian ini.

### Gunakan DynamoDB lokal langsung dalam kode

Seperti yang dijelaskan di bagian sebelumnya, Anda dapat mengatur `endpoint_url` langsung dalam kode untuk mengganti titik akhir dasar untuk menunjuk ke server DynamoDB lokal. Dalam kode Anda:

```
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);
```

[Contoh lengkap](#) tersedia di GitHub.

## Gunakan LocalStack menggunakan **config** file

Anda dapat mengatur [titik akhir khusus layanan](#) di file bersama Anda. AWS config Profil konfigurasi berikut ditetapkan `endpoint_url` untuk terhubung ke `localhost` pada port 4566. Untuk informasi selengkapnya tentang LocalStack konfigurasi, lihat [Mengakses LocalStack melalui titik akhir URL](#) di situs web LocalStack dokumen.

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

Ini SDK akan mengambil perubahan dalam config file bersama dan menerapkannya ke SDK klien Anda saat Anda menggunakan `localstack` profil. Dengan menggunakan pendekatan ini, kode Anda tidak perlu menyertakan referensi apa pun ke titik akhir, dan akan terlihat seperti:

```
// 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);
```

[Contoh lengkap](#) tersedia di GitHub.

## HTTP

The AWS SDK for Rust menyediakan konfigurasi HTTP terkait berikut:

Topik

- [Hiper 1.x](#)

## Hiper 1.x

### Note

Ini AWS SDK for Rust dikembangkan sebelum `hyper` merilis versi 1.0 yang stabil. SDK Saat ini default menggunakan versi v0.14 dari `hyper`

`hyper` adalah HTTP pustaka tingkat rendah untuk Rust yang dapat digunakan dengan AWS SDK for Rust untuk melakukan panggilan API layanan. Untuk dokumentasi referensi tentang peti ini, lihat [hyper - Rust](#). Secara default, SDK untuk Rust menggunakan versi v0.14 `hyper` dan penyedia `CryptoMode:Ring`

Klien dengan dukungan `hyper` 1.0 terletak di [aws-smithy-experimental](#) peti.

### Pilih penyedia kriptografi

Anda harus memilih `CryptoProvider` dan mengaktifkan fitur yang diperlukan di `aws-smithy-experimental` peti.

`CryptoProviderMode` berikut tersedia:


- `CryptoMode:AwsLc`- penyedia kriptografi berdasarkan [aws-lc-rs](#). Ini adalah perpustakaan kriptografi tujuan umum yang dikelola oleh tim AWS Kriptografi untuk AWS dan pelanggan mereka, berdasarkan kode dari proyek Google Boring dan SSL proyek Terbuka. SSL
  - Membutuhkan fitur `crypto-aws-lc`.
- `CryptoMode:AwsLcFips`- penyedia kriptografi FIPS yang sesuai berdasarkan [aws-lc-rs](#)
  - Membutuhkan fitur `crypto-aws-lc-fips`.

### Note

Ini mungkin memerlukan alat build tambahan untuk dapat dikompilasi. Untuk informasi lebih lanjut, rujuk instruksi [aws-lc-rs](#) repositori dan build.

- `CryptoMode:Ring`- penyedia kriptografi berdasarkan [ring](#). Ini adalah “Crypto kecil yang aman, cepat, menggunakan Rust”.

- Membutuhkan fitur `crypto-ring`.

 Note

Ini adalah penyedia krypto yang digunakan SDK untuk Rust di klien `hyper` v0.14 default.

Cara mengaktifkan fitur untuk peti

Contoh berikut menunjukkan cara menambahkan `crypto-aws-lc` fitur ke `aws-smithy-experimental` peti di proyek Anda: `Cargo.toml`

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

Atau, Anda dapat menjalankan perintah berikut pada prompt perintah di folder proyek Anda:

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

Untuk informasi selengkapnya, lihat [Fitur](#) di The Cargo Book.

## Buat klien **hyper** 1.0 dan gunakan dengan SDK

Untuk membuat klien `hyper` 1.0, di proyek Anda `Cargo.toml`, tambahkan `aws-smithy-experimental` peti. Yang `HyperClientBuilder` mengharuskan Anda untuk memilih `CryptoProvider`. Bagian sebelumnya membahas pilihan penyedia dan cara mengaktifkan salah satu fitur terkait krypto dari `petiaws-smithy-experimental`.

Kode berikut digunakan `CryptoMode::AwsLc` sebagai penyedia krypto untuk HTTP klien `hyper` 1.0 untuk digunakan dengan klien layanan Amazon Simple Storage Service (Amazon S3). Untuk contoh ini, proyek Anda `Cargo.toml` juga perlu menyertakan `peti,aws-sdk-s3`. Layanan AWS

```
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);
```

HTTPKlien dapat disetel pada struct konfigurasi bersama atau pada struct konfigurasi layanan tertentu.

## Pencegat

Anda dapat menggunakan pencegat untuk menghubungkan ke eksekusi API permintaan dan tanggapan. Pencegat adalah mekanisme terbuka di mana kode SDK panggilan yang Anda tulis untuk menyuntikkan perilaku ke dalam siklus hidup permintaan/respons. Dengan cara ini, Anda dapat mengubah permintaan dalam penerbangan, proses permintaan debug, kesalahan tampilan, dan lainnya.

Contoh berikut menunjukkan pencegat sederhana yang menambahkan header tambahan ke semua permintaan keluar sebelum loop coba lagi dimasukkan:

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

[Untuk informasi lebih lanjut dan kait pencegat yang tersedia, lihat sifat Intercept.](#)

## Pendaftaran pencegat

Anda mendaftarkan pencegat ketika Anda membangun klien layanan atau ketika Anda mengganti konfigurasi untuk operasi tertentu. Pendaftaran berbeda tergantung pada apakah Anda ingin pencegat berlaku untuk semua operasi untuk klien Anda atau hanya yang spesifik.

### Interceptor untuk semua operasi pada klien layanan

Untuk mendaftarkan pencegat untuk seluruh klien, tambahkan pencegat menggunakan pola. **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);
```

## Pencegat hanya untuk operasi tertentu

Untuk mendaftarkan pencegat hanya untuk satu operasi, gunakan ekstensi `customize`. Anda dapat mengganti konfigurasi klien layanan Anda di tingkat per operasi menggunakan metode ini. Untuk informasi selengkapnya tentang operasi yang dapat disesuaikan, lihat [Ganti konfigurasi operasi tunggal klien](#)

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

## Gunakan fitur observabilitas

Observabilitas adalah sejauh mana keadaan sistem saat ini dapat disimpulkan dari data yang dipancarkannya. Data yang dipancarkan biasanya disebut sebagai telemetri.

Topik

- [Aktifkan pencatatan AWS SDK for Rust kode](#)

## Aktifkan pencatatan AWS SDK for Rust kode

AWS SDK for Rust Menggunakan kerangka [penelusuran](#) untuk logging.

1. Dalam prompt perintah untuk direktori proyek Anda, tambahkan peti [tracing-subscriber](#) sebagai dependensi:

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

Ini menambahkan peti ke `[dependencies]` bagian `Cargo.toml` file Anda.

2. Inisialisasi pelanggan. Biasanya ini dilakukan di awal `main` fungsi sebelum memanggil apa pun SDK untuk operasi 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. Aktifkan logging menggunakan variabel RUST\_LOG lingkungan. Untuk mengaktifkan tampilan informasi logging, dalam prompt perintah, atur variabel RUST\_LOG lingkungan ke tingkat yang ingin Anda log. Contoh berikut menetapkan logging ke debug level:

#### Linux/macOS

```
$ RUST_LOG=debug
```

#### Windows

Jika Anda menggunakan VSCode, jendela terminal sering default ke PowerShell Verifikasi jenis prompt yang Anda gunakan.

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

#### PowerShell

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

4. Jalankan program:

```
$ cargo run
```

Anda akan melihat output tambahan di jendela konsol atau terminal.

Untuk informasi selengkapnya, lihat [memfilter peristiwa dengan variabel lingkungan](#) dari `tracing-subscriber` dokumentasi.

## Menafsirkan output log

Setelah Anda mengaktifkan logging dengan mengikuti langkah-langkah di bagian sebelumnya, informasi log tambahan akan dicetak ke standar secara default.

Jika Anda menggunakan format keluaran log default (disebut “penuh” oleh modul penelusuran), informasi yang Anda lihat di output log terlihat mirip dengan ini:

```
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
```

Setiap entri mencakup yang berikut:

- Stempel waktu entri log.
- Tingkat log entri. Ini adalah kata seperti `INFO`, `DEBUG`, atau `TRACE`.
- Kumpulan [bentang](#) bersarang dari mana entri log dihasilkan, dipisahkan oleh titik dua (":"). Ini membantu Anda mengidentifikasi sumber entri log.
- Jalur modul Rust yang berisi kode yang menghasilkan entri log.

- Teks pesan log.

Format output standar modul tracing menggunakan kode ANSI escape untuk mewarnai output. Ingatlah urutan pelarian ini saat memfilter atau mencari output.

#### Note

`sdk_invocation_id` yang muncul dalam kumpulan rentang bersarang adalah ID unik yang dihasilkan sisi klien oleh SDK untuk membantu mengkorelasikan pesan log. Ini tidak terkait dengan ID permintaan yang ditemukan dalam respons dari file Layanan AWS.

## Sempurnakan level logging Anda

Jika Anda menggunakan peti yang mendukung pemfilteran lingkungan, seperti `tracing_subscriber`, Anda dapat mengontrol verbositas log berdasarkan modul.

Anda dapat mengaktifkan level logging yang sama untuk setiap modul. Berikut ini menetapkan level logging `trace` untuk setiap modul:

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

Anda dapat mengaktifkan pencatatan tingkat penelusuran untuk modul tertentu. Dalam contoh berikut, hanya log yang berasal dari yang `aws_smithy_runtime` akan masuk pada `trace` level.

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

Anda dapat menentukan tingkat log yang berbeda untuk beberapa modul dengan memisahkannya dengan koma. Contoh berikut menyetel `aws_config` modul ke `trace` level logging, dan `aws_smithy_runtime` modul ke `debug` level logging.

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

Tabel berikut menjelaskan beberapa modul yang dapat Anda gunakan untuk memfilter pesan log:

Awalan	Deskripsi
<code>aws_smithy_runtime</code>	Permintaan dan respons pencatatan kawat

Awalan	Deskripsi
aws_config	Pemuatan kredensial
aws_sigv4	Permintaan penandatanganan dan permintaan kanonik

Salah satu cara untuk mengetahui modul mana yang perlu Anda sertakan dalam output log Anda adalah dengan terlebih dahulu mencatat semuanya, lalu temukan nama peti di output log untuk informasi yang Anda butuhkan. Kemudian Anda dapat mengatur variabel lingkungan yang sesuai dan menjalankan program Anda lagi.

## Ganti konfigurasi operasi tunggal klien

Setelah Anda [membuat klien layanan](#), konfigurasi menjadi tidak berubah dan akan berlaku untuk semua operasi selanjutnya. Meskipun konfigurasi tidak dapat dimodifikasi pada saat ini, konfigurasi dapat diganti berdasarkan per operasi.

Setiap pembuat operasi memiliki `customize` metode yang tersedia untuk membuat `CustomizableOperation` sehingga Anda dapat mengganti salinan individual dari konfigurasi yang ada. Konfigurasi klien asli akan tetap tidak dimodifikasi.

Contoh berikut menunjukkan pembuatan klien Amazon S3 yang memanggil dua operasi, yang kedua diganti untuk dikirim ke yang berbeda. Wilayah AWS Semua pemanggilan objek Amazon S3 menggunakan `us-east-1` wilayah kecuali saat API panggilan diganti secara eksplisit untuk menggunakan yang dimodifikasi. `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

Contoh sebelumnya adalah untuk Amazon S3, namun konsepnya sama untuk semua operasi. Operasi tertentu mungkin memiliki metode tambahan `CustomizableOperation`.

Untuk contoh menambahkan pengecat menggunakan `customize` untuk operasi tunggal, lihat [Pengecat hanya untuk operasi tertentu](#)

## Percobaan ulang

AWS SDK for Rust ini menyediakan perilaku coba ulang default dan opsi konfigurasi yang dapat disesuaikan. Panggilan untuk Layanan AWS sesekali mengembalikan pengecualian yang tidak terduga. Jenis kesalahan tertentu, seperti kesalahan pelambatan atau transien, mungkin berhasil jika panggilan dicoba ulang.

Perilaku coba lagi dapat dikonfigurasi secara global menggunakan variabel lingkungan atau pengaturan dalam AWS config file bersama. Untuk informasi tentang pendekatan ini, lihat [Coba lagi perilaku](#) di Panduan Referensi Alat AWS SDKs dan Alat. Ini juga mencakup informasi rinci tentang implementasi strategi coba lagi dan bagaimana memilih satu di atas yang lain.

Atau, opsi ini juga dapat dikonfigurasi dalam kode Anda, seperti yang ditunjukkan pada bagian berikut.

## Konfigurasi coba lagi default

Setiap klien layanan default ke konfigurasi strategi standard coba lagi yang disediakan melalui struct. [RetryConfig](#) Secara default, panggilan akan dicoba tiga kali (upaya awal, ditambah dua percobaan ulang). Selain itu, setiap percobaan ulang akan ditunda oleh durasi acak yang pendek untuk menghindari badai coba lagi. Konvensi ini cocok untuk sebagian besar kasus penggunaan tetapi mungkin tidak cocok dalam keadaan tertentu seperti sistem throughput tinggi.

Hanya beberapa jenis kesalahan yang dianggap dapat dicoba ulang oleh SDKs. Contoh kesalahan yang dapat dicoba ulang adalah:

- batas waktu socket
- pelambatan sisi layanan
- kesalahan layanan sementara seperti HTTP respons 5XX

Contoh-contoh berikut tidak dianggap dapat dicoba kembali:

- Parameter hilang atau tidak valid
- kesalahan otentikasi/keamanan
- pengecualian salah konfigurasi

Anda dapat menyesuaikan strategi standard coba lagi dengan mengatur upaya maksimum, penundaan, dan konfigurasi backoff.

## Upaya Maksimum

Anda dapat menyesuaikan upaya maksimum dalam kode Anda dengan memberikan modifikasi [RetryConfig](#) ke: `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;
```

## Penundaan dan backoff

Jika percobaan ulang diperlukan, strategi coba lagi default menunggu sebelum melakukan upaya berikutnya. Penundaan untuk percobaan ulang pertama kecil tetapi tumbuh secara eksponensial untuk percobaan ulang nanti. Jumlah penundaan maksimum dibatasi sehingga tidak tumbuh terlalu besar.

Jitter acak diterapkan pada penundaan di antara semua upaya. Jitter membantu mengurangi efek armada besar yang dapat menyebabkan badai coba lagi. Untuk diskusi yang lebih dalam tentang backoff eksponensial dan jitter, lihat [Exponential Backoff And Jitter](#) di [Blog Arsitektur.AWS](#)

Anda dapat menyesuaikan pengaturan penundaan dalam kode Anda dengan memberikan modifikasi [RetryConfig](#) ke kode Anda `aws_config::defaults`. Kode berikut menetapkan konfigurasi untuk menunda upaya percobaan ulang pertama hingga 100 milidetik dan jumlah waktu maksimum antara upaya coba lagi adalah 5 detik.

```
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;
```

## Mode Coba Ulang Adaptif

Sebagai alternatif dari strategi coba lagi standard mode, strategi coba lagi adaptive mode adalah pendekatan lanjutan yang mencari tingkat permintaan ideal untuk meminimalkan kesalahan pelambatan.

### Note

Coba ulang adaptif adalah mode coba lagi lanjutan. Menggunakan strategi ini biasanya tidak disarankan. Lihat [Perilaku Coba lagi](#) di Panduan Referensi Alat AWS SDKs dan.

Percobaan ulang adaptif mencakup semua fitur percobaan ulang standar. Ini menambahkan pembatas tingkat sisi klien yang mengukur tingkat permintaan yang dibatasi dibandingkan dengan permintaan non-throttled. Ini juga membatasi lalu lintas untuk mencoba tetap dalam bandwidth yang aman, idealnya menyebabkan kesalahan pelambatan nol.

Tarif beradaptasi secara real time untuk mengubah kondisi layanan dan pola lalu lintas dan dapat meningkatkan atau menurunkan tingkat lalu lintas yang sesuai. Secara kritis, pembatas tarif mungkin menunda upaya awal dalam skenario lalu lintas tinggi.

Anda dapat memilih strategi adaptive coba lagi dalam kode dengan menyediakan modifikasi:

### [RetryConfig](#)

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

## Timeout

AWS SDK for Rust ini menyediakan beberapa pengaturan untuk mengelola batas waktu permintaan dan aliran data yang terhenti. Ini membantu aplikasi Anda berperilaku optimal ketika penundaan dan kegagalan yang tidak terduga terjadi di jaringan.

### APIbatas waktu

Ketika ada masalah sementara yang dapat menyebabkan upaya permintaan memakan waktu lama atau gagal sepenuhnya, penting untuk meninjau dan mengatur batas waktu sehingga aplikasi Anda dapat gagal dengan cepat dan berperilaku optimal. Permintaan yang gagal dapat dicoba ulang secara otomatis oleh SDK ini adalah praktik yang baik untuk mengatur batas waktu untuk upaya individu dan seluruh permintaan.

The SDK for Rust menyediakan batas waktu default untuk membuat koneksi untuk permintaan. SDK tidak memiliki pengaturan waktu tunggu maksimum default untuk menerima respons untuk upaya permintaan atau untuk seluruh permintaan. Opsi batas waktu berikut tersedia:

Parameter	Nilai default	Deskripsi
Connect timeout	3,1 detik	Jumlah maksimum waktu untuk menunggu untuk membuat koneksi sebelum menyerah.



Parameter	Nilai default	Deskripsi
Batas waktu operasi	Tidak ada	Jumlah maksimum waktu untuk menunggu sebelum menerima respons dari SDK untuk Rust, termasuk semua percobaan ulang.
Batas waktu upaya operasi	Tidak ada	Jumlah maksimum waktu untuk menunggu satu HTTP upaya, setelah itu API panggilan dapat dicoba lagi.
Baca batas waktu	Tidak ada	Jumlah maksimum waktu untuk menunggu untuk membaca byte pertama respons dari saat permintaan dimulai.

Contoh berikut menunjukkan konfigurasi klien Amazon S3 dengan nilai batas waktu kustom:

```
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);
```

Ketika Anda menggunakan operasi dan mencoba timeout bersama-sama, Anda menetapkan batas keras pada total waktu yang dihabiskan untuk semua upaya di seluruh percobaan ulang. Anda juga menetapkan HTTP permintaan individu untuk gagal dengan cepat pada permintaan yang lambat.

Sebagai alternatif untuk menetapkan nilai batas waktu ini pada klien layanan untuk semua operasi, Anda dapat mengonfigurasi atau [menggantinya untuk satu permintaan](#).

### ⚠ Important

Batas waktu operasi dan upaya tidak berlaku untuk streaming data yang dikonsumsi setelah SDK for Rust mengembalikan respons. Sebagai contoh, mengonsumsi data dari `ByteStream` anggota respons tidak tunduk pada batas waktu operasi.

## Perlindungan aliran terhenti

The SDK for Rust menyediakan bentuk batas waktu lain yang terkait dengan mendeteksi aliran yang macet. Stream yang terhenti adalah aliran unggahan atau unduhan yang tidak menghasilkan data lebih lama dari masa tenggang yang dikonfigurasi. Ini membantu mencegah aplikasi menggantung tanpa batas waktu dan tidak pernah membuat kemajuan.

Perlindungan aliran yang terhenti akan mengembalikan kesalahan saat aliran menganggur lebih lama dari periode yang dapat diterima.

Secara default, SDK for Rust memungkinkan perlindungan streaming yang terhenti untuk unggahan dan unduhan dan mencari setidaknya 1 byte/detik aktivitas dengan masa tenggang 20 detik.

Contoh berikut menunjukkan penyesuaian `StalledStreamProtectionConfig` yang menonaktifkan perlindungan unggahan dan mengubah masa tenggang tanpa aktivitas menjadi 10 detik:

```
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**

Perlindungan aliran terhenti adalah opsi konfigurasi lanjutan. Kami merekomendasikan untuk mengubah nilai-nilai ini hanya jika aplikasi Anda membutuhkan kinerja yang lebih ketat atau jika itu menyebabkan beberapa masalah lain.

## Nonaktifkan perlindungan aliran yang macet

Contoh berikut menunjukkan cara menonaktifkan perlindungan aliran macet sepenuhnya:

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

# Gunakan AWS SDK for Rust

Pelajari cara-cara umum dan direkomendasikan menggunakan AWS SDK for Rust untuk bekerja dengan AWS layanan.

Topik

- [Membuat permintaan](#)
- [Praktik terbaik](#)
- [Konkurensi](#)
- [Penanganan kesalahan](#)
- [Buat fungsi Lambda](#)
- [Paginasi](#)
- [Buat presigned URLs](#)
- [Pengujian unit](#)
- [Pelayan](#)

## Membuat permintaan

Untuk membuat permintaan ke Layanan AWS, Anda harus terlebih dahulu [membuat klien layanan](#). Untuk setiap kode yang Layanan AWS Anda gunakan, ia memiliki peti sendiri dan miliknya sendiri `Client` untuk berinteraksi dengannya.

Ini `Client` mengekspos satu metode untuk setiap API operasi yang diekspos oleh layanan. Nilai kembali dari masing-masing metode ini adalah “pembangun lancar”, di mana input yang berbeda untuk itu API ditambahkan oleh rantai panggilan fungsi gaya pembangun. Setelah memanggil metode layanan, panggil `send()` untuk mendapatkan `Future` yang akan menghasilkan output yang berhasil atau `SdkError`. Untuk informasi selengkapnya tentang `SdkError`, lihat [Penanganan kesalahan](#).

Contoh berikut menunjukkan operasi dasar menggunakan Amazon S3 untuk membuat bucket di: us-west-2 Wilayah 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;
```

Setiap peti layanan memiliki modul tambahan yang digunakan untuk API input, seperti berikut ini:

- `types` Modul ini memiliki struct atau enum untuk memberikan informasi terstruktur yang lebih kompleks.
- `primitives` Modul ini memiliki tipe yang lebih sederhana untuk mewakili data seperti tanggal waktu atau gumpalan biner.

Lihat [dokumentasi API referensi](#) untuk peti layanan untuk organisasi dan informasi peti yang lebih rinci. Misalnya, `aws-sdk-s3` peti untuk Amazon Simple Storage Service memiliki beberapa [Modul](#). Dua di antaranya adalah:

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

## Praktik terbaik

Berikut ini adalah praktik terbaik untuk menggunakan AWS SDK for Rust.

### Gunakan kembali SDK klien jika memungkinkan

Bergantung pada bagaimana SDK klien dibangun, membuat klien baru dapat mengakibatkan setiap klien mempertahankan kumpulan HTTP koneksinya sendiri, cache identitas, dan sebagainya. Kami merekomendasikan berbagi klien atau setidaknya berbagi `SdkConfig` untuk menghindari biaya pembuatan sumber daya yang mahal. Semua SDK klien menerapkan `Clone` sebagai pembaruan jumlah referensi atom tunggal.

## Konfigurasi API batas waktu

SDK ini memberikan nilai default untuk beberapa opsi batas waktu, seperti batas waktu koneksi dan batas waktu socket, tetapi tidak untuk batas waktu API panggilan atau upaya panggilan individual. API ini adalah praktik yang baik untuk mengatur batas waktu untuk upaya individu dan seluruh permintaan. Ini akan memastikan aplikasi Anda gagal dengan cepat dengan cara yang optimal ketika ada masalah sementara yang dapat menyebabkan upaya permintaan membutuhkan waktu lebih lama untuk menyelesaikan atau masalah jaringan yang fatal.

Untuk informasi selengkapnya tentang mengonfigurasi batas waktu operasi, lihat [Timeout](#)

## Konkurensi

AWS SDK for Rust itu tidak memberikan kontrol konkurensi tetapi pengguna memiliki banyak opsi untuk mengimplementasikannya sendiri.

## Ketentuan

Istilah yang terkait dengan subjek ini mudah membingungkan dan beberapa istilah telah menjadi sinonim meskipun awalnya mewakili konsep yang terpisah. Dalam panduan ini, kami akan mendefinisikan yang berikut:

- **Tugas:** Beberapa “unit kerja” yang program Anda akan jalankan sampai selesai, atau mencoba untuk berjalan sampai selesai.
- **Sequential Computing:** Ketika beberapa tugas dijalankan satu demi satu.
- **Komputasi Bersamaan:** Ketika beberapa tugas dijalankan dalam periode waktu yang tumpang tindih.
- **Konkurensi:** Kemampuan komputer untuk menyelesaikan banyak tugas dalam urutan yang sewenang-wenang.
- **Multitasking:** Kemampuan komputer untuk menjalankan beberapa tugas secara bersamaan.
- **Kondisi Balapan:** Ketika perilaku program Anda berubah berdasarkan kapan tugas dimulai atau berapa lama waktu yang dibutuhkan untuk memproses tugas.
- **Konflik:** Konflik atas akses ke sumber daya bersama. Ketika dua atau lebih tugas ingin mengakses sumber daya pada saat yang sama, sumber daya itu “dalam pertengkaran”.
- **Deadlock:** Keadaan di mana tidak ada kemajuan lagi yang dapat dibuat. Ini biasanya terjadi karena dua tugas ingin memperoleh sumber daya masing-masing tetapi tidak ada tugas yang

akan melepaskan sumber daya mereka sampai sumber daya yang lain tersedia. Kebuntuan menyebabkan program menjadi sebagian atau seluruhnya tidak responsif.

## Contoh sederhana

Contoh pertama kami adalah program sekuensial. Dalam contoh selanjutnya, kita akan mengubah kode ini menggunakan teknik konkurensi. Contoh selanjutnya menggunakan kembali `build_client_and_list_objects_to_download()` metode yang sama dan membuat perubahan di `dalamnyamain()`.

Contoh tugas berikut adalah mengunduh semua file di bucket Amazon Simple Storage Service:

1. Mulailah dengan mencantumkan semua file. Simpan kunci dalam daftar.
2. Ulangi daftar, unduh setiap file secara bergantian

```
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

Dalam contoh ini, kami tidak akan menangani kesalahan, dan kami berasumsi bahwa bucket contoh tidak memiliki objek dengan kunci yang terlihat seperti jalur file. Dengan demikian, kita tidak akan membahas pembuatan direktori bersarang.

Karena arsitektur komputer modern, kita dapat menulis ulang program ini menjadi jauh lebih efisien. Kita akan melakukannya dalam contoh selanjutnya, tetapi pertama-tama, mari kita pelajari beberapa konsep lagi.

## Kepemilikan dan mutabilitas

Setiap nilai di Rust memiliki satu pemilik. Ketika pemilik keluar dari ruang lingkup, semua nilai yang dimilikinya juga akan dihapus. Pemilik dapat memberikan satu atau lebih referensi yang tidak dapat diubah ke nilai atau referensi tunggal yang dapat berubah. Kompiler Rust bertanggung jawab untuk memastikan bahwa tidak ada referensi yang hidup lebih lama dari pemiliknya.

Perencanaan dan desain tambahan diperlukan ketika beberapa tugas perlu mengakses sumber daya yang sama secara berubah-ubah. Dalam komputasi sekuensial, setiap tugas dapat mengakses sumber daya yang sama tanpa pertengkaran karena mereka berjalan satu demi satu secara berurutan. Namun, dalam komputasi bersamaan, tugas dapat berjalan dalam urutan apa pun, dan pada saat yang sama. Oleh karena itu, kita harus berbuat lebih banyak untuk membuktikan kepada



kompiler bahwa beberapa referensi yang dapat berubah tidak mungkin (atau setidaknya macet jika memang terjadi).

Pustaka standar Rust menyediakan banyak alat untuk membantu kami mencapai hal ini. Untuk informasi lebih lanjut tentang topik ini, lihat [Variabel dan Mutabilitas dan Memahami Kepemilikan](#) dalam buku *The Rust Programming Language*.

## Lebih banyak istilah!

Berikut ini adalah daftar “objek sinkronisasi”. Secara keseluruhan, mereka adalah alat yang diperlukan untuk meyakinkan kompiler bahwa program bersamaan kami tidak akan melanggar aturan kepemilikan.

### [Objek sinkronisasi perpustakaan standar:](#)

- [Busur](#): Penunjuk yang dimatikan R eferensi-C secara tomik. Ketika data dibungkus dalam sebuah `Arc`, itu dapat dibagikan secara bebas, tanpa khawatir pemilik tertentu menjatuhkan nilai lebih awal. Dalam pengertian ini, kepemilikan nilai menjadi “dibagikan”. Nilai dalam an `Arc` tidak dapat berubah, tetapi mungkin memiliki [mutabilitas interior](#).
- [Barrier](#): Memastikan beberapa thread akan menunggu satu sama lain untuk mencapai titik dalam program, sebelum melanjutkan eksekusi bersama-sama.
- [Condvar](#): a Cond ition Var iable menyediakan kemampuan untuk memblokir utas sambil menunggu peristiwa terjadi.
- [Mutex](#): mekanisme klusi Mut ual Ex yang memastikan bahwa paling banyak satu utas pada satu waktu dapat mengakses beberapa data. Secara umum, Mut ex kunci tidak boleh dipegang melintasi `.await` titik dalam kode.

### Objek [sinkronisasi Tokio](#):

Meskipun AWS SDKs dimaksudkan untuk menjadi `async -runtime-agnostik`, kami merekomendasikan penggunaan objek `tokio` sinkronisasi untuk kasus tertentu.

- [Mutex](#): Mirip dengan pustaka standar `Mutex`, tetapi dengan biaya yang sedikit lebih tinggi. Berbeda dengan standar `Mutex`, yang satu ini dapat disimpan di satu `.await` titik dalam kode.
- [Semaphore](#): Variabel yang digunakan untuk mengontrol akses ke sumber daya bersama dengan banyak tugas.

## Menulis ulang contoh kita menjadi lebih efisien (konkurensi ulir tunggal)

Dalam contoh modifikasi berikut, kita gunakan [futures\\_util::future::join\\_all](#) untuk menjalankan ALLget\_object permintaan secara bersamaan.

```
#[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;
}
```

Ini adalah cara paling sederhana untuk mendapatkan keuntungan dari konkurensi, tetapi juga memiliki beberapa masalah yang mungkin tidak jelas pada pandangan pertama:

1. Kami membuat semua input permintaan secara bersamaan. Jika kita tidak memiliki cukup memori untuk menyimpan semua input get\_object permintaan maka kita akan mengalami kesalahan alokasi out-of-memory "".
2. Kami membuat dan menunggu semua masa depan pada saat yang bersamaan. Amazon S3 membatasi permintaan jika kami mencoba mengunduh terlalu banyak sekaligus.

Untuk memperbaiki kedua masalah ini, kami harus membatasi jumlah permintaan yang kami kirim pada satu waktu. Kami akan melakukan ini dengan tokio [semaphore](#):

```
const CONCURRENCY_LIMIT: usize = 50;

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

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

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

Kami telah memperbaiki masalah penggunaan memori potensial dengan memindahkan pembuatan permintaan ke dalam async blok. Dengan cara ini, permintaan tidak akan dibuat sampai tiba waktunya untuk mengirimnya.

**Note**

Jika Anda memiliki memori untuk itu, mungkin lebih efisien untuk membuat semua input permintaan Anda sekaligus dan menyimpannya di memori sampai siap dikirim. Untuk mencoba ini, pindahkan pembuatan input permintaan di luar `async` blok.

Kami juga telah memperbaiki masalah pengiriman terlalu banyak permintaan sekaligus dengan membatasi permintaan dalam penerbangan `CONCURRENCY_LIMIT`.

**Note**

Nilai yang tepat `CONCURRENCY_LIMIT` untuk setiap proyek berbeda. Saat membuat dan mengirim permintaan Anda sendiri, cobalah untuk mengaturnya setinggi mungkin tanpa mengalami kesalahan pelambatan. Meskipun dimungkinkan untuk memperbarui batas konkurensi Anda secara dinamis berdasarkan rasio respons yang berhasil dan dibatasi yang dikirim kembali oleh layanan, itu berada di luar cakupan panduan ini karena kompleksitasnya.

## Menulis ulang contoh kita menjadi lebih efisien (konkurensi multi-utas)

Dalam dua contoh sebelumnya, kami melakukan permintaan kami secara bersamaan. Meskipun ini lebih efisien daripada menjalankannya secara sinkron, kita dapat membuat segalanya lebih efisien dengan menggunakan multi-threading. Untuk melakukan `initokio`, kita harus menelurkannya sebagai tugas terpisah.

**Note**

Contoh ini mengharuskan Anda menggunakan runtime `multi-threadedtokio`. Runtime ini terjaga keamanannya di belakang `rt-multi-thread` fitur. Dan, tentu saja, Anda harus menjalankan program Anda pada mesin `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;
}
```

Membagi pekerjaan menjadi tugas bisa jadi rumit. Melakukan I/O (input/output) biasanya memblokir. Runtime mungkin berjuang untuk menyeimbangkan kebutuhan tugas yang berjalan lama dengan tugas-tugas jangka pendek. Apa pun runtime yang Anda pilih, pastikan untuk membaca rekomendasi mereka untuk cara paling efisien untuk membagi pekerjaan Anda menjadi tugas. Untuk rekomendasi tokio runtime, lihat [Modul tokio::task](#).

## Mendebug aplikasi multi-utas

Tugas yang berjalan secara bersamaan dapat dijalankan dalam urutan apa pun. Dengan demikian, log program bersamaan bisa sangat sulit dibaca. Dalam SDK untuk Rust, kami sarankan menggunakan sistem `tracing` logging. Itu dapat mengelompokkan log dengan tugas khusus mereka, tidak peduli kapan mereka berjalan. Untuk panduan, lihat [Aktifkan pencatatan AWS SDK for Rust kode](#).

Alat yang sangat berguna untuk mengidentifikasi tugas yang telah terkunci adalah [tokio-console](#), yang merupakan alat diagnostik dan debugging untuk program Rust asinkron. Dengan menginstrumentasi dan menjalankan program Anda, dan kemudian menjalankan `tokio-console` aplikasi, Anda dapat melihat tampilan langsung dari tugas yang dijalankan program Anda. Tampilan ini mencakup informasi bermanfaat seperti jumlah waktu yang dihabiskan tugas untuk menunggu untuk memperoleh sumber daya bersama atau berapa kali tugas tersebut telah disurvei.

## Penanganan kesalahan

Memahami bagaimana dan kapan kesalahan AWS SDK for Rust pengembalian penting untuk membangun aplikasi berkualitas tinggi menggunakan SDK. Bagian berikut menjelaskan berbagai kesalahan yang mungkin Anda temui dari SDK dan cara menanganinya dengan tepat.

Setiap operasi mengembalikan `Result` tipe dengan jenis kesalahan disetel ke `SdkError<E, R = HttpResponse>`. `SdkError` adalah enum dengan beberapa jenis yang mungkin, yang disebut varian.

## Kesalahan layanan

Jenis kesalahan yang paling umum adalah `SdkError::ServiceError`. Kesalahan ini merupakan respons kesalahan dari file Layanan AWS. Misalnya, jika Anda mencoba mendapatkan objek dari Amazon S3 yang tidak ada, Amazon S3 mengembalikan respons kesalahan.

Ketika Anda menemukan sebuah `SdkError::ServiceError` itu berarti bahwa permintaan Anda berhasil dikirim ke Layanan AWS tetapi tidak dapat diproses. Ini bisa karena kesalahan dalam parameter permintaan atau karena masalah di sisi layanan.

Detail respons kesalahan disertakan dalam varian kesalahan. Contoh berikut menunjukkan cara mudah mendapatkan `ServiceError` varian yang mendasarinya dan menangani kasus kesalahan yang berbeda:

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

## Metadata kesalahan

Setiap kesalahan layanan memiliki metadata tambahan yang dapat diakses dengan mengimpor sifat khusus layanan.

- `<service>::error::ProvideErrorMetadataSifat` ini menyediakan akses ke kode kesalahan mentah yang tersedia dan pesan kesalahan yang dikembalikan dari layanan.
  - Untuk Amazon S3, sifat ini adalah. [aws\\_sdk\\_s3::error::ProvideErrorMetadata](#)

Anda juga bisa mendapatkan informasi yang mungkin berguna saat pemecahan masalah kesalahan layanan:

- `<service>::operation::RequestIdSifat` menambahkan metode ekstensi untuk mengambil ID AWS permintaan unik yang dihasilkan oleh layanan.
  - Untuk Amazon S3, sifat ini adalah. [aws\\_sdk\\_s3::operation::RequestId](#)

- `<service>::operation::RequestIdExt` menambahkan `extended_request_id()` metode untuk mendapatkan ID permintaan tambahan yang diperluas.
  - Hanya didukung oleh beberapa layanan.
  - Untuk Amazon S3, sifat ini adalah. [aws\\_sdk\\_s3::operation::RequestIdExt](#)

## Pencetakan kesalahan terperinci dengan **DisplayErrorContext**

Kesalahan pada SDK umumnya merupakan hasil dari rantai kegagalan seperti:

1. Mengirim permintaan gagal karena konektor mengembalikan kesalahan.
2. Konektor mengembalikan kesalahan karena penyedia kredensial mengembalikan kesalahan.
3. Penyedia kredensial mengembalikan kesalahan karena disebut layanan dan layanan itu mengembalikan kesalahan.
4. Layanan mengembalikan kesalahan karena permintaan kredensial tidak memiliki otorisasi yang benar.

Secara default, tampilan kesalahan ini hanya menghasilkan “kegagalan pengiriman”. Ini tidak memiliki detail yang membantu memecahkan masalah kesalahan. The SDK for Rust menyediakan reporter kesalahan sederhana yang disebut `DisplayErrorContext`.

- `<service>::error::DisplayErrorContextStruct` menambahkan fungsionalitas untuk menampilkan konteks kesalahan penuh.
  - Untuk Amazon S3, struct ini. [aws\\_sdk\\_s3::error::DisplayErrorContext](#)

Saat kami membungkus kesalahan yang akan ditampilkan dan mencetaknya, `DisplayErrorContext` berikan pesan yang jauh lebih rinci mirip dengan yang berikut ini:

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



```

        ServiceError {
            source: UnauthorizedException(
                UnauthorizedException {
                    message: Some("Session token not found or
invalid"),
                    meta: ErrorMetadata {
                        code: Some("UnauthorizedException"),
                        message: Some("Session token not found
or invalid"),
                        extras: Some({"aws_request_id":
"1b6d7476-f5ec-4a16-9890-7684ccee7d01"})
                    }
                },
            raw: Response {
                status: StatusCode(401),
                headers: Headers {
                    headers: {
                        "date": HeaderValue { _private:
H0("Thu, 04 Jul 2024 07:41:21 GMT") },
                        "content-type": HeaderValue { _private:
H0("application/json") },
                        { _private: H0("114") },
                        "access-control-expose-headers":
HeaderValue { _private: H0("RequestId") },
                        "access-control-expose-headers":
HeaderValue { _private: H0("x-amzn-RequestId") },
                        "requestid": HeaderValue { _private:
H0("1b6d7476-f5ec-4a16-9890-7684ccee7d01") },
                        "server": HeaderValue { _private:
H0("AWS SS0") },
                        "x-amzn-requestid": HeaderValue
{ _private: H0("1b6d7476-f5ec-4a16-9890-7684ccee7d01") }
                    }
                },
                body: SdkBody {
                    inner: Once(
                        Some(
                            b"{
                    \"message\": \"Session token not
found or invalid\",
                    \"__type\":
                    \"com.amazonaws.switchboard.portal#UnauthorizedException\"}"

```

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

## Buat fungsi Lambda

Untuk dokumentasi terperinci tentang pengembangan AWS Lambda fungsi dengan AWS SDK for Rust, lihat [Membangun fungsi Lambda dengan Rust di Panduan AWS Lambda Pengembang](#). Dokumentasi itu memandu Anda menggunakan:

- Peti klien runtime Rust Lambda untuk fungsionalitas inti, [aws-lambda-rust-runtime](#)
- [Alat baris perintah yang direkomendasikan untuk menerapkan biner fungsi Rust ke Lambda dengan Cargo Lambda](#).

Selain contoh terpandu yang ada di Panduan AWS Lambda Pengembang, ada juga contoh kalkulator Lambda yang tersedia di [Repositori Contoh AWS SDK Kode](#) di GitHub

## Paginasi

Banyak AWS operasi mengembalikan hasil terpotong ketika muatan terlalu besar untuk dikembalikan dalam satu respons. Sebagai gantinya, layanan mengembalikan sebagian data dan token untuk mengambil set item berikutnya. Pola ini dikenal sebagai pagination.

AWS SDK for Rust Termasuk metode ekstensi `into_paginator` pada pembuat operasi yang dapat digunakan untuk memberi halaman hasil secara otomatis untuk Anda. Anda hanya perlu menulis kode yang memproses hasilnya. Semua pembuat operasi pagination memiliki `into_paginator()` metode yang tersedia yang mengekspos a [PaginatorStream<Item>](#) to paginate atas hasil.

- Di Amazon S3, salah satu contohnya adalah.

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

Contoh berikut menggunakan Amazon Simple Storage Service. Namun, konsepnya sama untuk layanan apa pun yang memiliki satu atau lebih paginasi APIs.

Contoh kode berikut menunjukkan contoh paling sederhana yang menggunakan [try\\_collect\(\)](#) metode untuk mengumpulkan semua hasil paginasi menjadi `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<_>>();
```

Terkadang, Anda ingin memiliki kontrol lebih besar atas paging dan tidak menarik semuanya ke dalam memori sekaligus. Contoh berikut mengulangi objek dalam bucket Amazon S3 hingga tidak ada lagi.

```
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);
    }
}
```

## Buat presigned URLs

Anda dapat melakukan presign permintaan untuk beberapa AWS API operasi sehingga pemanggil lain dapat menggunakan permintaan nanti tanpa menunjukkan kredensialnya sendiri.

Misalnya, asumsikan bahwa Jane memiliki akses ke objek Amazon Simple Storage Service (Amazon S3) dan dia ingin sementara berbagi akses objek dengan Alejandro. Jane dapat membuat `GetObject` permintaan yang telah ditetapkan sebelumnya untuk dibagikan dengan Alejandro sehingga ia dapat mengunduh objek tanpa memerlukan akses ke kredensial Jane atau memiliki miliknya sendiri. Kredensial yang digunakan oleh presigned URL adalah milik Jane karena dia adalah AWS pengguna yang menghasilkan file. URL

Untuk mempelajari selengkapnya tentang presigned URLs di Amazon S3, [lihat Bekerja dengan URLs presigned](#) di Panduan Pengguna Layanan Penyimpanan Sederhana Amazon.

## Dasar-dasar pra-penandatanganan

AWS SDK for Rust ini menyediakan `presigned()` metode pada operasi pembangun fasih yang dapat digunakan untuk mendapatkan permintaan yang telah ditetapkan sebelumnya.

Contoh berikut membuat `GetObject` permintaan yang telah ditetapkan sebelumnya untuk Amazon S3. Permintaan ini berlaku selama 5 menit setelah pembuatan.

```
use std::time::Duration;
use aws_config::BehaviorVersion;
use aws_sdk_s3::presigning::PresigningConfig;
```

```

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

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

let presigned = s3.get_object()
    .presigned(
        PresigningConfig::builder()
            .expires_in(Duration::from_secs(60 * 5))
            .build()
            .expect("less than one week")
    )
    .await?;

```

`presigned()` Metode mengembalikan `aResult<PresignedRequest, SdkError<E, R>>`.

Yang dikembalikan `PresignedRequest` berisi metode untuk mendapatkan komponen HTTP permintaan termasuk metode, URI, dan header apa pun. Semua ini perlu dikirim ke layanan, jika ada, agar permintaan valid. Banyak permintaan yang telah ditetapkan sebelumnya dapat diwakili oleh URI satu-satunya.

## Presigning **POST** dan permintaan **PUT**

Banyak operasi yang dapat dipresignable hanya memerlukan a URL dan harus dikirim sebagai HTTP GET permintaan. Beberapa operasi, bagaimanapun, mengambil tubuh dan harus dikirim sebagai HTTP POST atau HTTP PUT permintaan bersama dengan header dalam beberapa kasus. Penandatanganan permintaan ini identik dengan permintaan GET presigning, tetapi memanggil permintaan yang telah ditetapkan sebelumnya lebih rumit.

Berikut ini adalah contoh penandatanganan permintaan Amazon `PutObject` S3 dan mengubahnya menjadi permintaan [http::request::Request](#) yang dapat dikirim menggunakan klien HTTP yang Anda pilih.

Untuk menggunakan `into_http_1x_request()` metode ini, tambahkan `http-1x` fitur ke `aws-sdk-s3` peti Anda di `Cargo.toml` file Anda:

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

File sumber:

```
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);
```

## Penandatanganan Mandiri

### Note

Ini adalah kasus penggunaan lanjutan. Ini tidak diperlukan atau direkomendasikan untuk sebagian besar pengguna.

Ada beberapa kasus penggunaan di mana perlu untuk membuat permintaan yang ditandatangani di luar konteks SDK for Rust. Untuk itu Anda dapat menggunakan [aws-sigv4](#) peti secara independen dari SDK.

Berikut ini adalah contoh untuk mendemonstrasikan elemen dasar, lihat dokumentasi peti untuk lebih jelasnya.

Tambahkan `aws-sigv4` dan `http` peti ke `Cargo.toml` file Anda:

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

File sumber:

```
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);
```

## Pengujian unit

Meskipun ada banyak cara Anda dapat menerapkan pengujian unit dalam AWS SDK for Rust proyek Anda, ada beberapa yang kami rekomendasikan:

- Gunakan [automock](#) dari [mockall](#) peti untuk membuat dan menjalankan pengujian Anda.
- Gunakan runtime AWS Smithy [StaticReplayClient](#) untuk membuat HTTP klien palsu yang dapat digunakan sebagai pengganti HTTP klien standar yang biasanya digunakan oleh. Layanan

AWS Klien ini mengembalikan HTTP respons yang Anda tentukan daripada berkomunikasi dengan layanan melalui jaringan, sehingga pengujian mendapatkan data yang diketahui untuk tujuan pengujian.

## Secara otomatis menghasilkan tiruan menggunakan mockall

Anda dapat secara otomatis menghasilkan sebagian besar implementasi tiruan yang dibutuhkan pengujian Anda dengan menggunakan yang populer [automock](#) dari peti [mockall](#).

Contoh ini menguji metode kustom yang disebut `determine_prefix_file_size()`. Metode ini memanggil metode `list_objects()` pembungkus khusus yang memanggil Amazon S3. Dengan mengejek `list_objects()`, `determine_prefix_file_size()` metode ini dapat diuji tanpa benar-benar menghubungi Amazon S3.

1. Dalam prompt perintah untuk direktori proyek Anda, tambahkan [mockall](#) peti sebagai dependensi:

```
$ cargo add mockall
```

Ini menambahkan peti ke `[dependencies]` bagian `Cargo.toml` file Anda.

2. Sertakan `automock` modul dari `mockall` peti.

Sertakan juga perpustakaan lain yang terkait dengan Layanan AWS yang Anda uji, dalam hal ini, Amazon S3.

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

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

3. Selanjutnya, tambahkan kode yang menentukan mana dari dua implementasi struktur pembungkus Amazon S3 aplikasi yang akan digunakan.
  - Yang asli ditulis untuk mengakses Amazon S3 melalui jaringan.
  - Implementasi tiruan yang dihasilkan oleh `mockall`.

Dalam contoh ini, yang dipilih diberi nama `S3`. Seleksi bersyarat berdasarkan `test` atribut:



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

4. `S3ImplStruct` adalah implementasi struktur pembungkus Amazon S3 yang benar-benar mengirimkan permintaan ke AWS
- Saat pengujian diaktifkan, kode ini tidak digunakan karena permintaan dikirim ke tiruan dan bukan AWS. `dead_code` atribut memberitahu linter untuk tidak melaporkan masalah jika `S3Impl` tipe tidak digunakan.
  - Kondisional `#[cfg_attr(test, automock)]` menunjukkan bahwa ketika pengujian diaktifkan, `automock` atribut harus diatur. Ini memberitahu `mockall` untuk menghasilkan tiruan `S3Impl` yang akan diberi nama `MockS3Impl`.
  - Dalam contoh ini, `list_objects()` metodenya adalah panggilan yang ingin Anda ejek. `automock` secara otomatis membuat `expect_list_objects()` metode untuk Anda.

```
#[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. Buat fungsi pengujian dalam modul bernama `test`.

- Kondisional `#[cfg(test)]` menunjukkan bahwa `mockall` harus membangun modul pengujian jika `test` atributnya `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);
}
}

```

- Setiap tes menggunakan `let mut mock = MockS3Impl::default();` untuk membuat mock instance dari `MockS3Impl`.

- Ini menggunakan `expect_list_objects()` metode tiruan (yang dibuat secara otomatis oleh `automock`) untuk mengatur hasil yang diharapkan ketika `list_objects()` metode digunakan di tempat lain dalam kode.
  - Setelah harapan ditetapkan, ia menggunakan ini untuk menguji fungsi dengan `determine_prefix_file_size()`. Nilai yang dikembalikan diperiksa untuk mengonfirmasi bahwa itu benar, menggunakan pernyataan.
6. `determine_prefix_file_size()` Fungsi ini menggunakan pembungkus Amazon S3 untuk mendapatkan ukuran file awalan:

```
#[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)
}
```

Tipe S3 ini digunakan untuk memanggil fungsi wrapped SDK for Rust untuk mendukung keduanya `S3Impl` dan `MockS3Impl` saat membuat HTTP permintaan. Mock yang dihasilkan secara otomatis oleh `mockall` melaporkan kegagalan pengujian apa pun saat pengujian diaktifkan.

Anda dapat [melihat kode lengkap untuk contoh-contoh ini](#) di GitHub.

## Simulasikan HTTP lalu lintas menggunakan pemutaran ulang statis

`aws-smithy-runtime` termasuk kelas utilitas uji yang disebut [StaticReplayClient](#). Kelas HTTP klien ini dapat ditentukan bukan HTTP klien default saat membuat Layanan AWS objek.

Saat menginisialisasi `StaticReplayClient`, Anda memberikan daftar pasangan HTTP permintaan dan respons sebagai `ReplayEvent` objek. Saat pengujian sedang berjalan, setiap HTTP permintaan dicatat dan klien mengembalikan HTTP respons berikutnya yang ditemukan `ReplayEvent` di daftar acara berikutnya sebagai respons HTTP klien. Ini memungkinkan pengujian dijalankan menggunakan data yang diketahui dan tanpa koneksi jaringan.

### Menggunakan replay statis

Untuk menggunakan pemutaran ulang statis, Anda tidak perlu menggunakan pembungkus. Sebagai gantinya, tentukan seperti apa lalu lintas jaringan yang sebenarnya untuk data yang akan digunakan pengujian Anda, dan berikan data lalu lintas itu `StaticReplayClient` kepada yang akan digunakan setiap kali SDK masalah permintaan dari Layanan AWS klien.

#### Note

Ada beberapa cara untuk mengumpulkan lalu lintas jaringan yang diharapkan, termasuk AWS CLI dan banyak penganalisis lalu lintas jaringan dan alat packet sniffer.

- Buat daftar `ReplayEvent` objek yang menentukan HTTP permintaan yang diharapkan dan tanggapan yang harus dikembalikan untuk mereka.
- Buat `StaticReplayClient` menggunakan daftar HTTP transaksi yang dibuat pada langkah sebelumnya.
- Buat objek konfigurasi untuk AWS klien, menentukan `StaticReplayClient` sebagai `Config` objek. `http_client`
- Buat objek Layanan AWS klien, menggunakan konfigurasi yang dibuat pada langkah sebelumnya.

- Lakukan operasi yang ingin Anda uji, menggunakan objek layanan yang dikonfigurasi untuk menggunakan `StaticReplayClient`. Setiap kali SDK mengirim API permintaan ke AWS, respons berikutnya dalam daftar digunakan.

#### Note

Respons berikutnya dalam daftar selalu dikembalikan, bahkan jika permintaan yang dikirim tidak cocok dengan yang ada di vektor `ReplayEvent` objek.

- Ketika semua permintaan yang diinginkan telah dibuat, panggil `StaticReplayClient.assert_requests_match()` fungsi untuk memverifikasi bahwa permintaan yang dikirim oleh SDK cocok dengan yang ada dalam daftar `ReplayEvent` objek.

## Contoh

Mari kita lihat tes untuk `determine_prefix_file_size()` fungsi yang sama pada contoh sebelumnya, tetapi menggunakan pemutaran ulang statis alih-alih mengejek.

1. Dalam prompt perintah untuk direktori proyek Anda, tambahkan [aws-smithy-runtime](#) peti sebagai dependensi:

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

Ini menambahkan peti ke `[dependencies]` bagian `Cargo.toml` file Anda.

2. Dalam file sumber Anda, sertakan `aws_smithy_runtime` jenis yang Anda perlukan.

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

3. Tes dimulai dengan membuat `ReplayEvent` struktur yang mewakili setiap HTTP transaksi yang harus dilakukan selama pengujian. Setiap peristiwa berisi objek HTTP permintaan dan objek HTTP respons yang mewakili informasi yang biasanya Layanan AWS akan dibalas. Peristiwa ini diteruskan ke panggilan ke `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]);

```

Hasilnya disimpan di `replay_client`. Ini merupakan HTTP klien yang kemudian dapat digunakan oleh SDK for Rust dengan menentukannya dalam konfigurasi klien.

4. Untuk membuat klien Amazon S3, panggil `from_conf()` fungsi kelas klien untuk membuat klien menggunakan objek konfigurasi:

```

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(),
);

```

Objek konfigurasi ditentukan menggunakan `http_client()` metode pembangun, dan kredensialnya ditentukan menggunakan metode `credentials_provider()` Kredensialnya

dibuat menggunakan fungsi yang disebut `make_s3_test_credentials()`, yang mengembalikan struktur kredensial palsu:

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

Kredensi ini tidak harus valid karena tidak akan benar-benar dikirim ke. AWS

5. Jalankan pengujian dengan memanggil fungsi yang membutuhkan pengujian. Dalam contoh ini, nama fungsi itu adalah `determine_prefix_file_size()`. Parameter pertamanya adalah objek klien Amazon S3 yang akan digunakan untuk permintaannya. Oleh karena itu, tentukan klien yang dibuat menggunakan permintaan `StaticReplayClient` so ditangani oleh itu daripada keluar melalui jaringan:

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

assert_eq!(19, size);

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

Ketika panggilan ke `determine_prefix_file_size()` selesai, pernyataan digunakan untuk mengonfirmasi bahwa nilai yang dikembalikan cocok dengan nilai yang diharapkan. Kemudian, `assert_requests_match()` fungsi `StaticReplayClient` metode disebut. Fungsi ini memindai HTTP permintaan yang direkam dan mengonfirmasi bahwa semuanya cocok dengan yang ditentukan dalam array `ReplayEvent` objek yang disediakan saat membuat klien replay.

Anda dapat [melihat kode lengkap untuk contoh-contoh ini](#) di GitHub.



# Pelayan

Pelayan adalah abstraksi sisi klien yang digunakan untuk polling sumber daya sampai keadaan yang diinginkan tercapai, atau sampai ditentukan bahwa sumber daya tidak akan memasuki keadaan yang diinginkan. Ini adalah tugas umum ketika bekerja dengan layanan yang pada akhirnya konsisten, seperti Amazon Simple Storage Service, atau layanan yang membuat sumber daya secara asinkron, seperti Amazon Elastic Compute Cloud. Menulis logika untuk terus polling status sumber daya dapat menjadi rumit dan rawan kesalahan. Tujuan pelayan adalah untuk memindahkan tanggung jawab ini dari kode pelanggan dan ke dalam AWS SDK for Rust, yang memiliki pengetahuan mendalam tentang aspek waktu untuk operasi. AWS

Layanan AWS yang memberikan dukungan untuk pelayan termasuk modul. `<service>::waiters`

- `<service>::client::Waiters` Sifat tersebut menyediakan metode pelayan untuk klien. Metode diimplementasikan untuk Client struct. Semua metode pelayan mengikuti konvensi penamaan standar `wait_until_<Condition>`
  - Untuk Amazon S3, sifat ini adalah. [aws\\_sdk\\_s3::client::Waiters](#)

Contoh berikut menggunakan Amazon S3. Namun, konsepnya sama untuk setiap Layanan AWS yang memiliki satu atau lebih pelayan didefinisikan.

Contoh kode berikut menunjukkan penggunaan fungsi pelayan alih-alih menulis logika polling untuk menunggu bucket ada setelah dibuat.

```
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

Setiap metode tunggu mengembalikan a `Result<FinalPoll<...>, WaiterError<...>>` yang dapat digunakan untuk mendapatkan respons akhir dari mencapai kondisi yang diinginkan atau kesalahan. Lihat [FinalPoll](#) dan [WaiterError](#) di API dokumentasi Rust untuk detailnya.

# SDK untuk contoh kode Rust

Contoh kode dalam topik ini menunjukkan cara menggunakan AWS SDK for Rust with AWS.

Dasar-dasar adalah contoh kode yang menunjukkan kepada Anda bagaimana melakukan operasi penting dalam suatu layanan.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Skenario adalah contoh kode yang menunjukkan kepada Anda bagaimana menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan atau dikombinasikan dengan yang lain Layanan AWS.

Beberapa layanan berisi kategori contoh tambahan yang menunjukkan cara memanfaatkan pustaka atau fungsi khusus untuk layanan.

## Layanan

- [API Contoh gateway menggunakan SDK untuk Rust](#)
- [API API Contoh Manajemen Gateway menggunakan SDK untuk Rust](#)
- [Contoh Application Auto Scaling menggunakan SDK untuk Rust](#)
- [Contoh Aurora menggunakan SDK untuk Rust](#)
- [Contoh Auto Scaling menggunakan SDK untuk Rust](#)
- [Contoh Amazon Bedrock Runtime menggunakan SDK untuk Rust](#)
- [Contoh Penyedia Identitas Amazon Cognito menggunakan SDK untuk Rust](#)
- [Contoh Amazon Cognito Sync menggunakan SDK untuk Rust](#)
- [Contoh Firehose yang digunakan SDK untuk Rust](#)
- [Contoh Amazon DocumentDB menggunakan untuk Rust SDK](#)
- [Contoh DynamoDB menggunakan untuk Rust SDK](#)
- [EBS Contoh Amazon menggunakan SDK untuk Rust](#)
- [EC2 Contoh Amazon menggunakan SDK untuk Rust](#)
- [ECR Contoh Amazon menggunakan SDK untuk Rust](#)

- [ECSContoh Amazon menggunakan SDK untuk Rust](#)
- [EKSContoh Amazon menggunakan SDK untuk Rust](#)
- [AWS Glue contoh menggunakan SDK untuk Rust](#)
- [IAMcontoh menggunakan SDK untuk Rust](#)
- [AWS IoT contoh menggunakan SDK untuk Rust](#)
- [Contoh Kinesis yang digunakan SDK untuk Rust](#)
- [AWS KMS contoh menggunakan SDK untuk Rust](#)
- [Contoh Lambda menggunakan SDK untuk Rust](#)
- [MediaLive contoh menggunakan SDK untuk Rust](#)
- [MediaPackage contoh menggunakan SDK untuk Rust](#)
- [Contoh Amazon Polly menggunakan SDK untuk Rust](#)
- [QLDBcontoh menggunakan SDK untuk Rust](#)
- [RDSContoh Amazon menggunakan SDK untuk Rust](#)
- [Contoh Layanan RDS Data Amazon menggunakan SDK untuk Rust](#)
- [Contoh Amazon Rekognition menggunakan untuk Rust SDK](#)
- [Route 53 contoh menggunakan SDK untuk Rust](#)
- [Contoh Amazon S3 menggunakan SDK untuk Rust](#)
- [SageMaker Contoh AI menggunakan SDK untuk Rust](#)
- [Contoh Secrets Manager menggunakan SDK untuk Rust](#)
- [Amazon SES API v2 contoh menggunakan SDK untuk Rust](#)
- [SNSContoh Amazon menggunakan SDK untuk Rust](#)
- [SQSContoh Amazon menggunakan SDK untuk Rust](#)
- [AWS STS contoh menggunakan SDK untuk Rust](#)
- [Contoh Systems Manager menggunakan SDK untuk Rust](#)
- [Contoh Amazon Transcribe menggunakan SDK untuk Rust](#)

## APIContoh gateway menggunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust with API Gateway.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Skenario adalah contoh kode yang menunjukkan kepada Anda bagaimana menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan atau dikombinasikan dengan yang lain Layanan AWS.

AWS kontribusi komunitas adalah contoh yang dibuat dan dikelola oleh banyak tim AWS. Untuk memberikan umpan balik, gunakan mekanisme yang disediakan di repositori terkait.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)
- [Skenario](#)
- [AWS kontribusi komunitas](#)

## Tindakan

### GetRestApis

Contoh kode berikut menunjukkan cara menggunakan `GetRestApis`.

SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

Menampilkan Amazon API Gateway REST APIs di Wilayah.

```
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(())
}
```

- Untuk API detailnya, lihat [GetRestApis AWSSDK](#) untuk API referensi Rust.

## Skenario

Membuat aplikasi nirserver untuk mengelola foto

Contoh kode berikut menunjukkan cara membuat aplikasi tanpa server yang memungkinkan pengguna mengelola foto menggunakan label.

SDK untuk Rust

Menunjukkan cara mengembangkan aplikasi manajemen aset foto yang mendeteksi label dalam gambar menggunakan Amazon Rekognition dan menyimpannya untuk pengambilan nanti.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Untuk mendalami tentang asal usul contoh ini, lihat postingan di [Komunitas AWS](#).

Layanan yang digunakan dalam contoh ini

- API Gerbang
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3

- Amazon SNS

## AWS kontribusi komunitas

Membangun dan menguji aplikasi tanpa server

Contoh kode berikut menunjukkan cara membangun dan menguji aplikasi tanpa server menggunakan API Gateway dengan Lambda dan DynamoDB

SDK untuk Rust

Menunjukkan cara membangun dan menguji aplikasi tanpa server yang terdiri dari API Gateway dengan Lambda dan DynamoDB menggunakan Rust. SDK

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda

## API Gateway Contoh Manajemen menggunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust with API Gateway Management API.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)

# Tindakan

## PostToConnection

Contoh kode berikut menunjukkan cara menggunakan `PostToConnection`.

### SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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);
```



- Untuk API detailnya, lihat [PostToConnection AWS SDK untuk API referensi Rust](#).

## Contoh Application Auto Scaling menggunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust with Application Auto Scaling.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)

## Tindakan

### **DescribeScalingPolicies**

Contoh kode berikut menunjukkan cara menggunakan `DescribeScalingPolicies`.

SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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!("{:?}", policy);
}
println!("Next token: {:?}", response.next_token());

Ok(())
}
```

- Untuk API detailnya, lihat [DescribeScalingPolicies AWS SDK untuk API referensi Rust](#).

## Contoh Aurora menggunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust with Aurora.

Dasar-dasar adalah contoh kode yang menunjukkan kepada Anda bagaimana melakukan operasi penting dalam suatu layanan.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

Memulai

Halo Aurora

Contoh kode berikut ini menunjukkan cara mulai menggunakan Aurora.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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_identifier().unwrap_or("Unknown");
        let class = cluster.db_cluster_instance_class().unwrap_or("Unknown");
        println!("\tDatabase: {name},");
        println!("\t Engine: {engine},");
        println!("\t      ID: {id},");
        println!("\tInstance: {class},");
    }

    Ok(())
}

```

- Untuk API detailnya, lihat [D escribeDBClusters AWS](#) SDK untuk API referensi Rust.

## Topik

- [Hal-hal mendasar](#)
- [Tindakan](#)

## Hal-hal mendasar

Pelajari dasar-dasarnya

Contoh kode berikut ini menunjukkan cara:

- Membuat grup parameter klaster DB Aurora dan mengatur nilai parameter.
- Membuat klaster DB yang menggunakan grup parameter.
- Membuat instans DB yang berisi basis data.
- Mengambil snapshot klaster DB, lalu membersihkan sumber daya.

### SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

Pustaka yang berisi fungsi khusus skenario untuk skenario 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_idenfier
            .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_idenfier
            .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;

```

## Pengujian untuk perpustakaan menggunakan automock di sekitar pembungkus RDS Klien.

```

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_idenfier = 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");
}

```

Biner untuk menjalankan skenario dari depan ke ujung, menggunakan inquirer sehingga pengguna dapat membuat beberapa keputusan.

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

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

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

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

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

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

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

```

}

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

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

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

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

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

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

```

```

    }

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

    Ok(scenario)
}

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

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

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

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

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

```

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

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

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

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

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

    Ok(())
}
```

```

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

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

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

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

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

    Ok(())
}

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

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

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

```



```

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

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

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

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

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

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

```

```

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

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

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

```

Pembungkus di sekitar RDS layanan Amazon yang memungkinkan otomocking untuk pengujian.

```

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_identifier(name)
        .db_cluster_parameter_group_name(parameter_group)
        .engine(engine)
        .engine_version(version)
        .master_username(username)
        .master_user_password(password.expose_secret())
        .send()
        .await
}

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

```
        .await
    }

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

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

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

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

```
}

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

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

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

Cargo.toml dengan dependensi yang digunakan dalam skenario ini.



```
[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"] }
```

- Untuk API detailnya, lihat topik berikut AWS SDK untuk API referensi Rust.
  - [C reateDBCluster](#)
  - [C reateDBCluster ParameterGroup](#)
  - [reateDBCluster Cuplikan C](#)
  - [C reateDBInstance](#)
  - [D eleteDBCluster](#)
  - [D eleteDBCluster ParameterGroup](#)
  - [D eleteDBInstance](#)
  - [D escribeDBCluster ParameterGroups](#)
  - [D escribeDBCluster Parameter](#)
  - [D escribeDBCluster Cuplikan](#)

- [DescribeDBClusters](#)
- [describeDBEngineVersi D](#)
- [DescribeDBInstances](#)
- [DescribeOrderableDBInstanceOptions](#)
- [ModifyDBCluster ParameterGroup](#)

## Tindakan

### CreateDBCluster

Contoh kode berikut menunjukkan cara menggunakan CreateDBCluster.

#### SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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_idenfifier = create_db_instance
    .unwrap()
    .db_instance
    .and_then(|i| i.db_instance_idenfifier);

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

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

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

```
        ));
    }

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

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

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

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

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

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

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

```

    version: &str,
    username: &str,
    password: SecretString,
) -> Result<CreateDbClusterOutput, SdkError<CreateDBClusterError>> {
    self.inner
        .create_db_cluster()
        .db_cluster_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;
}

```

- Untuk API detailnya, lihat [C reateDBCluster AWS](#) SDK untuk API referensi Rust.

## CreateDBClusterParameterGroup

Contoh kode berikut menunjukkan cara menggunakan `CreateDBClusterParameterGroup`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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());
}

```

- Untuk API detailnya, lihat [C reateDBCluster ParameterGroup AWS](#) SDK untuk API referensi Rust.

## CreateDBClusterSnapshot

Contoh kode berikut menunjukkan cara menggunakan `CreateDBClusterSnapshot`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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;
}

```

- Untuk API detailnya, lihat [C createDBCluster Snapshot AWS](#) SDK untuk API referensi Rust.

## CreateDBInstance

Contoh kode berikut menunjukkan cara menggunakan `CreateDBInstance`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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;
}
```

```
        .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_idenfifier(cluster_name)
        .db_instance_idenfifier(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_idenfifier(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;
}

```

- Untuk API detailnya, lihat [C reateDBInstance AWS](#) SDK untuk API referensi Rust.

## DeleteDBCluster

Contoh kode berikut menunjukkan cara menggunakan DeleteDBCluster.

### SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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;
}

```

- Untuk API detailnya, lihat [DeleteDBCluster AWS SDK](#) untuk API referensi Rust.

## DeleteDBClusterParameterGroup

Contoh kode berikut menunjukkan cara menggunakan `DeleteDBClusterParameterGroup`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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_parameter_group(
    &self,

```

```

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

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

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

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

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

    mock_rds

```

```

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

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

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

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

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

```

```

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

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

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

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

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))

```

```

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

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

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

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

tokio::time::pause();

```

```

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

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

```

- Untuk API detailnya, lihat [DeleteDBCluster ParameterGroup AWS](#) SDK untuk API referensi Rust.

## DeleteDBInstance

Contoh kode berikut menunjukkan cara menggunakan `DeleteDBInstance`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(

```





```
    }

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

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

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

tokio::time::pause();
let assertions = tokio::spawn(async move {

```

```

        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });

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

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

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

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(

```

```

        ErrorKind::Other,
        "describe db instances error",
    )),
    Response::new(StatusCode::try_from(400).unwrap(), SdkBody::empty()),
    ))
});

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

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

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

```

```

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

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

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

```

- Untuk API detailnya, lihat [DeleteDBInstance AWS SDK](#) untuk API referensi Rust.

## DescribeDBClusterParameters

Contoh kode berikut menunjukkan cara menggunakan `DescribeDBClusterParameters`.

## SDKuntuk Rust

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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");
}

```

- Untuk API detailnya, lihat [DescribeDBCluster Parameter AWS](#) SDK untuk API referensi Rust.

## DescribeDBClusters

Contoh kode berikut menunjukkan cara menggunakan `DescribeDBClusters`.

## SDKuntuk Rust

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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;
}

```

- Untuk API detailnya, lihat [D escribeDBClusters AWS](#) SDK untuk API referensi Rust.

## DescribeDBEngineVersions

Contoh kode berikut menunjukkan cara menggunakan `DescribeDBEngineVersions`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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"
    );
}

```

- Untuk API detailnya, lihat [describeDBEngineVersi D AWS SDK](#) untuk API referensi Rust.

## DescribeDBInstances

Contoh kode berikut menunjukkan cara menggunakan `DescribeDBInstances`.

## SDKuntuk Rust

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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 describe_db_instances(
        &self,
    ) -> Result<DescribeDbInstancesOutput, SdkError<DescribeDBInstancesError>> {
        self.inner.describe_db_instances().send().await
    }

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

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

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()

```

```

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

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

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

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

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

```

```

        .db_cluster_parameter_group_name("MockParamGroup")
        .build(),
    );

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

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

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

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

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_idenfier("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;
}

```

- Untuk API detailnya, lihat [D escribeDBInstances AWS](#) SDK untuk API referensi Rust.

## DescribeOrderableDBInstanceOptions

Contoh kode berikut menunjukkan cara menggunakan `DescribeOrderableDBInstanceOptions`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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"
    );
}

```

- Untuk API detailnya, lihat [DescribeOrderableDBInstanceOptions AWSSDK](#) untuk API referensi Rust.

## ModifyDBClusterParameterGroup

Contoh kode berikut menunjukkan cara menggunakan `ModifyDBClusterParameterGroup`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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");
}
```

- Untuk API detailnya, lihat [ModifyDBCluster ParameterGroup AWS SDK](#) untuk API referensi Rust.

## Contoh Auto Scaling menggunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust with Auto Scaling.

Dasar-dasar adalah contoh kode yang menunjukkan kepada Anda bagaimana melakukan operasi penting dalam suatu layanan.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

Memulai

Halo Auto Scaling

Contoh kode berikut menunjukkan cara memulai menggunakan Auto Scaling.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [DescribeAutoScalingGroups AWSSDK](#) untuk API referensi Rust.

## Topik

- [Hal-hal mendasar](#)
- [Tindakan](#)

## Hal-hal mendasar

### Pelajari dasar-dasarnya

Contoh kode berikut ini menunjukkan cara:

- Buat grup EC2 Auto Scaling Amazon dengan template peluncuran dan Availability Zone, dan dapatkan informasi tentang menjalankan instans.
- Aktifkan pengumpulan CloudWatch metrik Amazon.
- Perbarui kapasitas yang diinginkan grup dan tunggu instance dimulai.
- Mengakhiri sebuah instance dalam grup.

- Buat daftar aktivitas penskalaan yang terjadi sebagai respons terhadap permintaan pengguna dan perubahan kapasitas.
- Dapatkan statistik untuk CloudWatch metrik, lalu bersihkan sumber daya.

## SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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
        ))?;

        Ok(())
    }
}
```

```

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

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

        writeln!(f, "\t\t\t\t\t Activities:");
        match &self.activities {
            Ok(activities) => {
                for activity in activities {
                    writeln!(
                        f,
                        "\t\t\t\t\t- {} Progress: {}% Status: {:?} End: {:?}"
                        ,
                        activity.cause().unwrap_or("Unknown"),
                        activity.progress.unwrap_or(-1),
                        activity.status_code(),
                        // activity.status_message().unwrap_or_default()
                        activity.end_time(),
                    )?;
                }
            }
            Err(e) => writeln!(f, "\t\t\t\t\t! {e}")?,
        }
    }
}

```

```
    }

    Ok(())
}

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

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

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

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

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



```

    }
}

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

impl Error for ScenarioError {
    // While `Error` can capture `source` information about the underlying error,
    // for this example
    // the ScenarioError captures the underlying information in MetadataError and
    // treats it as a
    // single Error from this Crate. In other contexts, it may be appropriate to
    // model the error
    // as including the SdkError as its source.
}

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

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

        let auto_scaling_group_name = String::from(AUTOSCALING_GROUP_NAME);

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

        let availability_zones: Vec<String> = availability_zones

```

```

        .unwrap()
        .availability_zones
        .unwrap_or_default()
        .iter()
        .take(3)
        .map(|z| z.zone_name.clone().unwrap())
        .collect();

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

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

    // 2. CreateAutoScalingGroup: pass it the launch template you created in
    step 0. Give it min/max of 1 instance.

```

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

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

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

```

        &err,
    ));
}
return Err(errs);
}

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

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

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

}

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

```

```

        .delete_auto_scaling_group()
        .auto_scaling_group_name(self.auto_scaling_group_name.clone())
        .send()
        .await;

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

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

if early_exit.is_err() {
    early_exit
} else {
    // Wait for delete_group to finish
    let waiter = Waiter::new();
    let mut errors = Vec::<ScenarioError>::new();
    while errors.len() < 3 {
        if let Err(e) = waiter.sleep().await {
            errors.push(e);
            continue;
        }
        let describe_group = self
            .autoscaling

```

```

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

pub async fn describe_scenario(&self) -> AutoScalingScenarioDescription {
    let group = self
        .autoscaling
        .describe_auto_scaling_groups()
        .auto_scaling_group_names(self.auto_scaling_group_name.clone())
        .send()
        .await
        .map(|s| {
            s.auto_scaling_groups()
                .iter()
                .map(|s| {

```

```

        format!(
            "{}: {}",
            s.auto_scaling_group_name().unwrap_or("Unknown"),
            s.status().unwrap_or("Unknown")
        )
    })
    .collect::<Vec<String>>()
})
.map_err(|e| {
    ScenarioError::new("Failed to describe auto scaling groups for
scenario", &e)
});

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

// 10. DescribeScalingActivities: list the scaling activities that have
occurred for the group.
// Bonus: use CloudWatch API to get and show some metrics collected for
the group.
// CW.ListMetrics with Namespace='AWS/AutoScaling' and
Dimensions=[{'Name': 'AutoScalingGroupName', 'Value': }]
// CW.GetMetricStatistics with Statistics='Sum'. Start and End times must
be in UTC!
let activities = self
    .autoscaling
    .describe_scaling_activities()
    .auto_scaling_group_name(self.auto_scaling_group_name.clone())
    .into_paginator()
    .items()
    .send()
    .collect::<Result<Vec<_>, _>>()
    .await
    .map_err(|e| {
        anyhow!(
            "There was an error retrieving scaling activities: {}",
            DisplayErrorContext(&e)
        )
    });

AutoScalingScenarioDescription {
    group,

```

```
        instances,
        activities,
    }
}

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

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

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

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

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

pub async fn wait_for_no_scaling(&self) -> Result<(), ScenarioError> {
    let waiter = Waiter::new();
    let mut scaling = true;
    while scaling {
```



```

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

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

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

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

    Ok(())
}

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

```

```

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

    pub async fn scale_min_size(&self, size: i32) -> Result<(), ScenarioError> {
        let update_group = self
            .autoscaling
            .update_auto_scaling_group()
            .auto_scaling_group_name(self.auto_scaling_group_name.clone())
            .min_size(size)
            .send()
            .await;
        if let Err(err) = update_group {
            return Err(ScenarioError::new(
                format!("Failer to update group to min size ({size}))").as_str(),
                &err,
            ));
        }
        Ok(())
    }

    pub async fn scale_max_size(&self, size: i32) -> Result<(), ScenarioError> {
        // 5. UpdateAutoScalingGroup: update max size to 3.

```

```

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

pub async fn scale_desired_capacity(&self, capacity: i32) -> Result<(),
ScenarioError> {
    // 7. SetDesiredCapacity: set desired capacity to 2.
    // Wait for a second instance to launch.
    let update_group = self
        .autoscaling
        .set_desired_capacity()
        .auto_scaling_group_name(self.auto_scaling_group_name.clone())
        .desired_capacity(capacity)
        .send()
        .await;
    if let Err(err) = update_group {
        return Err(ScenarioError::new(
            format!("Failed to update group to desired capacity
({{capacity}})").as_str(),
            &err,
        ));
    }
    Ok(())
}

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

```

```

        .send()
        .await;

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

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

    Ok(())
}

pub async fn terminate_some_instance(&self) -> Result<(), ScenarioError> {
    // Retrieve a list of instances in the auto scaling group.
    let auto_scaling_group = self.get_group().await?;
    let instances = auto_scaling_group.instances();
    // Or use other logic to find an instance to terminate.
    let instance = instances.first();
    if let Some(instance) = instance {
        let instance_id = if let Some(instance_id) = instance.instance_id() {
            instance_id
        } else {
            return Err(ScenarioError::with("Missing instance id"));
        };
        let termination = self

```

```
        .ec2
        .terminate_instances()
        .instance_ids(instance_id)
        .send()
        .await;
    if let Err(err) = termination {
        Err(ScenarioError::new(
            "There was a problem terminating an instance",
            &err,
        ))
    } else {
        Ok(())
    }
} else {
    Err(ScenarioError::with("There was no instance to terminate"))
}
}
}

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

- Untuk API detailnya, lihat topik berikut AWS SDK untuk API referensi Rust.
  - [CreateAutoScalingGroup](#)
  - [DeleteAutoScalingGroup](#)
  - [DescribeAutoScalingGroups](#)
  - [DescribeAutoScalingInstances](#)
  - [DescribeScalingActivities](#)
  - [DisableMetricsCollection](#)
  - [EnableMetricsCollection](#)
  - [SetDesiredCapacity](#)
  - [TerminateInstanceInAutoScalingGroup](#)
  - [UpdateAutoScalingGroup](#)

## Tindakan

### CreateAutoScalingGroup

Contoh kode berikut menunjukkan cara menggunakan `CreateAutoScalingGroup`.

SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [CreateAutoScalingGroup AWSSDK untuk API referensi Rust](#).

### DeleteAutoScalingGroup

Contoh kode berikut menunjukkan cara menggunakan `DeleteAutoScalingGroup`.

## SDKuntuk Rust

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [DeleteAutoScalingGroup AWSSDKuntuk API referensi Rust](#).

## DescribeAutoScalingGroups

Contoh kode berikut menunjukkan cara menggunakan `DescribeAutoScalingGroups`.

## SDKuntuk Rust

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}

```

- Untuk API detailnya, lihat [DescribeAutoScalingGroups AWS SDK](#) untuk API referensi Rust.

## DescribeAutoScalingInstances

Contoh kode berikut menunjukkan cara menggunakan `DescribeAutoScalingInstances`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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))
    }

```

- Untuk API detailnya, lihat [DescribeAutoScalingInstances AWSSDK](#) untuk API referensi Rust.

## DescribeScalingActivities

Contoh kode berikut menunjukkan cara menggunakan `DescribeScalingActivities`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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::

```

- Untuk API detailnya, lihat [DescribeScalingActivities AWSSDK](#) untuk API referensi Rust.

## DisableMetricsCollection

Contoh kode berikut menunjukkan cara menggunakan `DisableMetricsCollection`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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;

```

- Untuk API detailnya, lihat [DisableMetricsCollection AWS SDK untuk API referensi Rust](#).

## EnableMetricsCollection

Contoh kode berikut menunjukkan cara menggunakan `EnableMetricsCollection`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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;
```

- Untuk API detailnya, lihat [EnableMetricsCollection AWS SDK untuk API referensi Rust](#).

## SetDesiredCapacity

Contoh kode berikut menunjukkan cara menggunakan `SetDesiredCapacity`.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [SetDesiredCapacity AWS SDKuntuk API referensi Rust](#).

## TerminateInstanceInAutoScalingGroup

Contoh kode berikut menunjukkan cara menggunakan `TerminateInstanceInAutoScalingGroup`.

## SDKuntuk Rust

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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"))
    }
}

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

```

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

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

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

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

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

```

- Untuk API detailnya, lihat [TerminateInstanceInAutoScalingGroup AWS SDK untuk API referensi Rust](#).

## UpdateAutoScalingGroup

Contoh kode berikut menunjukkan cara menggunakan UpdateAutoScalingGroup.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [UpdateAutoScalingGroup AWS SDKuntuk API referensi Rust](#).

## Contoh Amazon Bedrock Runtime menggunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan Runtime AWS SDK for Rust with Amazon Bedrock.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

### Topik

- [Antropik Claude](#)



# Antropik Claude

## Bercakap

Contoh kode berikut menunjukkan cara mengirim pesan teks ke Anthropic Claude, menggunakan Bedrock Converse. API

## SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode AWS](#).

Kirim pesan teks ke Anthropic Claude, menggunakan Bedrock's Converse. API

```
#[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)
}

```

Gunakan pernyataan, utilitas Kesalahan, dan konstanta.

```

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",
        })
    }
}
}

```

- Untuk API detailnya, lihat [Converse](#) in AWS SDK untuk API referensi Rust.

## ConverseStream

Contoh kode berikut menunjukkan cara mengirim pesan teks ke Anthropic Claude, menggunakan Bedrock Converse API dan memproses aliran respons secara real-time.

## SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode AWS](#).

Kirim pesan teks ke Anthropic Claude dan streaming token balasan, menggunakan Bedrock.

## ConverseStream API

```
#[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(),
    })
}

```

Gunakan pernyataan, utilitas Kesalahan, dan konstanta.

```

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(),
        ),
    },
}
}
}

```

- Untuk API detailnya, lihat [ConverseStream AWS SDK](#) untuk API referensi Rust.

### Skenario: Penggunaan alat dengan Converse API

Contoh kode berikut menunjukkan bagaimana membangun interaksi khas antara aplikasi, model AI generatif, dan alat yang terhubung atau APIs untuk memediasi interaksi antara AI dan dunia luar. Ini menggunakan contoh menghubungkan cuaca eksternal API ke model AI sehingga dapat memberikan informasi cuaca real-time berdasarkan input pengguna.

### SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

Skenario utama dan logika untuk demo. Ini mengatur percakapan antara pengguna, Amazon Bedrock Converse API, dan alat cuaca.

```

#[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();
}
}

```

Alat cuaca yang digunakan oleh demo. Script ini mendefinisikan spesifikasi alat dan mengimplementasikan logika untuk mengambil data cuaca menggunakan dari Open-Meteo. API

```

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

```

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

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

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

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

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

```

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

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

```

### Utilitas untuk mencetak Blok Konten Pesan.

```

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:?})"
        )))
    }
}
}

```

### Gunakan pernyataan, utilitas Kesalahan, dan konstanta.

```

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(),
        })
    }
}
}

```

- Untuk API detailnya, lihat [Converse](#) in AWS SDK untuk API referensi Rust.

## Contoh Penyedia Identitas Amazon Cognito menggunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust with Amazon Cognito Identity Provider.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)

### Tindakan

#### ListUserPools

Contoh kode berikut menunjukkan cara menggunakan `ListUserPools`.



## SDKuntuk Rust

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [ListUserPools AWSSDKuntuk API referensi Rust](#).

## Contoh Amazon Cognito Sync menggunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust with Amazon Cognito Sync.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)

## Tindakan

### ListIdentityPoolUsage

Contoh kode berikut menunjukkan cara menggunakan ListIdentityPoolUsage.

SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [ListIdentityPoolUsage AWSSDK](#) untuk API referensi Rust.

## Contoh Firehose yang digunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust with Firehose.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)

## Tindakan

### **PutRecordBatch**

Contoh kode berikut menunjukkan cara menggunakan `PutRecordBatch`.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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
}
```

- Untuk API detailnya, lihat [PutRecordBatch AWS SDKuntuk API referensi Rust](#).

## Contoh Amazon DocumentDB menggunakan untuk Rust SDK

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust dengan Amazon DocumentDB.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

### Topik

- [Contoh nirserver](#)

## Contoh nirserver

Memanggil fungsi Lambda dari pemicu Amazon DocumentDB

Contoh kode berikut menunjukkan cara menerapkan fungsi Lambda yang menerima peristiwa yang dipicu dengan menerima catatan dari aliran perubahan DocumentDB. Fungsi mengambil payload DocumentDB dan mencatat isi catatan.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori [contoh Nirserver](#).

Mengonsumsi acara Amazon DocumentDB dengan Lambda menggunakan Rust.

```
use lambda_runtime::{service_fn, tracing, Error, LambdaEvent};
use aws_lambda_events::{
    event::documentdb::{DocumentDbEvent, DocumentDbInnerEvent},
};

// Built with the following dependencies:
//lambda_runtime = "0.11.1"
//serde_json = "1.0"
//tokio = { version = "1", features = ["macros"] }
//tracing = { version = "0.1", features = ["log"] }
//tracing-subscriber = { version = "0.3", default-features = false, features =
    ["fmt"] }
//aws_lambda_events = "0.15.0"

async fn function_handler(event: LambdaEvent<DocumentDbEvent>) ->Result<(), Error> {

    tracing::info!("Event Source ARN: {:?}", event.payload.event_source_arn);
    tracing::info!("Event Source: {:?}", event.payload.event_source);

    let records = &event.payload.events;

    if records.is_empty() {
```

```
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }

    for record in records{
        log_document_db_event(record);
    }

    tracing::info!("Document db records processed");

    // Prepare the response
    Ok(())
}

fn log_document_db_event(record: &DocumentDbInnerEvent)-> Result<(), Error>{
    tracing::info!("Change Event: {:?}" , record.event);

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();

    let func = service_fn(function_handler);
    lambda_runtime::run(func).await?;
    Ok(())
}
```

## Contoh DynamoDB menggunakan untuk Rust SDK

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust dengan DynamoDB.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Skenario adalah contoh kode yang menunjukkan kepada Anda bagaimana menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan atau dikombinasikan dengan yang lain Layanan AWS.

AWS kontribusi komunitas adalah contoh yang dibuat dan dikelola oleh banyak tim AWS. Untuk memberikan umpan balik, gunakan mekanisme yang disediakan di repositori terkait.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)
- [Skenario](#)
- [Contoh nirserver](#)
- [AWS kontribusi komunitas](#)

## Tindakan

### CreateTable

Contoh kode berikut menunjukkan cara menggunakan CreateTable.

SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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))
    }
}
}
```



- Untuk API detailnya, lihat [CreateTable AWSSDK](#) untuk API referensi Rust.

## DeleteItem

Contoh kode berikut menunjukkan cara menggunakan `DeleteItem`.

### SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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)),
    }
}
```

- Untuk API detailnya, lihat [DeleteItem AWSSDK](#) untuk API referensi Rust.

## DeleteTable

Contoh kode berikut menunjukkan cara menggunakan `DeleteTable`.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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())),
    }
}
```

- Untuk API detailnya, lihat [DeleteTable AWSSDKuntuk API referensi Rust](#).

## ListTables

Contoh kode berikut menunjukkan cara menggunakanListTables.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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)
}
```

Tentukan apakah tabel ada.

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

- Untuk API detailnya, lihat [ListTables AWSSDK](#) untuk API referensi Rust.

## PutItem

Contoh kode berikut menunjukkan cara menggunakan `PutItem`.

### SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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,
})
}
```

- Untuk API detailnya, lihat [PutItem AWSSDK](#) untuk API referensi Rust.

## Query

Contoh kode berikut menunjukkan cara menggunakan `Query`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode AWS](#).

Temukan film yang dibuat pada tahun tertentu.

```
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![])
    }
}
```

- Untuk API detailnya, lihat [Kueri AWS](#) SDK untuk API referensi Rust.

## Scan

Contoh kode berikut menunjukkan cara menggunakan `Scan`.

## SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [Memindai API](#) referensi Rust.AWS SDK

## Skenario

### Connect ke instans lokal

Contoh kode berikut menunjukkan cara mengganti titik akhir untuk terhubung URL ke penyebaran pengembangan lokal DynamoDB dan file. AWS SDK

Untuk informasi selengkapnya, lihat [DynamoDB Local](#).

## SDKuntuk Rust

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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:?}"),
    }
}
```

## Membuat aplikasi nirserver untuk mengelola foto

Contoh kode berikut menunjukkan cara membuat aplikasi tanpa server yang memungkinkan pengguna mengelola foto menggunakan label.

### SDK untuk Rust

Menunjukkan cara mengembangkan aplikasi manajemen aset foto yang mendeteksi label dalam gambar menggunakan Amazon Rekognition dan menyimpannya untuk pengambilan nanti.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Untuk mendalami tentang asal usul contoh ini, lihat postingan di [Komunitas AWS](#).

Layanan yang digunakan dalam contoh ini

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

Melakukan kueri tabel menggunakan PartiQL

Contoh kode berikut ini menunjukkan cara:

- Dapatkan item dengan menjalankan SELECT pernyataan.
- Tambahkan item dengan menjalankan INSERT pernyataan.
- Perbarui item dengan menjalankan UPDATE pernyataan.
- Hapus item dengan menjalankan DELETE pernyataan.



## SDK untuk Rust

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(())
}
}

```

- Untuk API detailnya, lihat [ExecuteStatement AWS SDK untuk API referensi Rust](#).

Simpan EXIF dan informasi gambar lainnya

Contoh kode berikut ini menunjukkan cara:

- Dapatkan EXIF informasi dariJPG,JPEG, atau PNG file.
- Mengunggah file gambar ke bucket Amazon S3.
- Menggunakan Amazon Rekognition untuk mengidentifikasi tiga atribut teratas (label) dalam file.
- Tambahkan informasi EXIF dan label ke tabel Amazon DynamoDB di Wilayah.

## SDKuntuk Rust

Mendapatkan EXIF informasi dariJPG,JPEG, atau PNG file, upload file gambar ke bucket Amazon S3, gunakan Amazon Rekognition untuk mengidentifikasi tiga atribut teratas (label di Amazon Rekognition) dalam file, dan menambahkan informasi dan EXIF label ke tabel Amazon DynamoDB di Wilayah.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Amazon Rekognition
- Amazon S3

## Contoh nirserver

Memanggil fungsi Lambda dari pemicu DynamoDB

Contoh kode berikut menunjukkan bagaimana menerapkan fungsi Lambda yang menerima peristiwa yang dipicu oleh menerima catatan dari aliran DynamoDB. Fungsi mengambil payload DynamoDB dan mencatat isi catatan.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori [contoh Nirserver](#).

Mengonsumsi acara DynamoDB dengan Lambda menggunakan 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(())
}
```

Melaporkan kegagalan item batch untuk fungsi Lambda dengan pemicu DynamoDB

Contoh kode berikut menunjukkan cara mengimplementasikan respons batch sebagian untuk fungsi Lambda yang menerima peristiwa dari aliran DynamoDB. Fungsi melaporkan kegagalan item batch dalam respons, memberi sinyal ke Lambda untuk mencoba lagi pesan tersebut nanti.

SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori [contoh Nirserver](#).

Melaporkan kegagalan item batch DynamoDB dengan Lambda menggunakan 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 kontribusi komunitas

### Membangun dan menguji aplikasi tanpa server

Contoh kode berikut menunjukkan cara membangun dan menguji aplikasi tanpa server menggunakan API Gateway dengan Lambda dan DynamoDB

### SDK untuk Rust

Menunjukkan cara membangun dan menguji aplikasi tanpa server yang terdiri dari API Gateway dengan Lambda dan DynamoDB menggunakan Rust. SDK

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda



# EBSContoh Amazon menggunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust dengan AmazonEBS.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)

## Tindakan

### CompleteSnapshot

Contoh kode berikut menunjukkan cara menggunakanCompleteSnapshot.

SDKuntuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [CompleteSnapshot AWSSDK](#) untuk API referensi Rust.

## PutSnapshotBlock

Contoh kode berikut menunjukkan cara menggunakan `PutSnapshotBlock`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [PutSnapshotBlock AWSSDK](#) untuk API referensi Rust.

## StartSnapshot

Contoh kode berikut menunjukkan cara menggunakan `StartSnapshot`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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())
}
```

- Untuk API detailnya, lihat [StartSnapshot AWS SDK](#) untuk API referensi Rust.

## EC2 Contoh Amazon menggunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust dengan Amazon EC2.

Dasar-dasar adalah contoh kode yang menunjukkan kepada Anda bagaimana melakukan operasi penting dalam suatu layanan.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.


Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

Memulai

Halo Amazon EC2

Contoh kode berikut menunjukkan cara memulai menggunakan AmazonEC2.

SDK untuk Rust

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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}");
    }
}
```

- Untuk API detailnya, lihat [DescribeSecurityGroups AWSSDK](#) untuk API referensi Rust.

## Topik

- [Hal-hal mendasar](#)
- [Tindakan](#)

## Hal-hal mendasar

Pelajari dasar-dasarnya

Contoh kode berikut ini menunjukkan cara:

- Membuat pasangan kunci dan grup keamanan.
- Pilih Amazon Machine Image (AMI) dan jenis instans yang kompatibel, lalu buat instance.
- Menghentikan dan memulai ulang instans.
- Kaitkan alamat IP Elastis dengan instans Anda.
- Connect ke instans Anda dengan SSH, lalu bersihkan sumber daya.

## SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

EC2InstanceScenarioImplementasinya berisi logika untuk menjalankan contoh secara keseluruhan.

```

    //! 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!
    ("-----");
}

```

EC2ImplStruct berfungsi sebagai titik automock untuk pengujian, dan fungsinya membungkus panggilan. 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)
    }
}
}

```

SSMStruct berfungsi sebagai titik automock untuk pengujian, dan fungsinya membungkus SSM SDK panggilan.

```

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)
    }
}

```

Skenario ini menggunakan beberapa struct gaya “Manajer” untuk menangani akses ke sumber daya yang dibuat dan dihapus di seluruh skenario.

```

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."),
        }
    }
}
}

```

Titik masuk utama untuk skenario.

```
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;
}
```

- Untuk API detailnya, lihat topik berikut AWS SDK untuk API referensi Rust.

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

- [StartInstances](#)
- [StopInstances](#)
- [TerminateInstances](#)
- [UnmonitorInstances](#)

## Tindakan

### AllocateAddress

Contoh kode berikut menunjukkan cara menggunakan `AllocateAddress`.

SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

- Untuk API detailnya, lihat [AllocateAddress AWS SDK untuk API referensi Rust](#).

### AssociateAddress

Contoh kode berikut menunjukkan cara menggunakan `AssociateAddress`.

## SDKuntuk Rust

**Note**

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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)
}
```

- Untuk API detailnya, lihat [AssociateAddress AWSSDKuntuk API referensi Rust](#).

**AuthorizeSecurityGroupIngress**

Contoh kode berikut menunjukkan cara menggunakan `AuthorizeSecurityGroupIngress`.

## SDKuntuk Rust

**Note**

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}

```

- Untuk API detailnya, lihat [AuthorizeSecurityGroupIngress AWSSDK untuk API referensi Rust](#).

## CreateKeyPair

Contoh kode berikut menunjukkan cara menggunakan `CreateKeyPair`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode AWS](#).



Implementasi Rust yang memanggil `create_key_pair` EC2 Klien dan mengekstrak materi yang dikembalikan.

```
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))
}
```

Fungsi yang memanggil `create_key` impl dan menyimpan kunci pribadi dengan aman. 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)
    }

```

- Untuk API detailnya, lihat [CreateKeyPair AWSSDK](#) untuk API referensi Rust.

## CreateSecurityGroup

Contoh kode berikut menunjukkan cara menggunakan `CreateSecurityGroup`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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)
}

```

- Untuk API detailnya, lihat [CreateSecurityGroup AWSSDKuntuk API referensi Rust](#).

## CreateTags

Contoh kode berikut menunjukkan cara menggunakan `CreateTags`.

SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

Contoh ini menerapkan tag Nama Setelah membuat instance.

```

pub async fn create_instance<'a>(
    &self,
    image_id: &'a str,
    instance_type: InstanceType,
    key_pair: &'a KeyPairInfo,
    security_groups: Vec<&'a SecurityGroup>,
) -> Result<String, EC2Error> {
    let run_instances = self
        .client

```

```

        .run_instances()
        .image_id(image_id)
        .instance_type(instance_type)
        .key_name(
            key_pair
                .key_name()
                .ok_or_else(|| EC2Error::new("Missing key name when launching
instance"))?),
        )
        .set_security_group_ids(Some(
            security_groups
                .iter()
                .filter_map(|sg| sg.group_id.clone())
                .collect(),
        ))
        .min_count(1)
        .max_count(1)
        .send()
        .await?;

if run_instances.instances().is_empty() {
    return Err(EC2Error::new("Failed to create instance"));
}

let instance_id = run_instances.instances()[0].instance_id().unwrap();
let response = self
    .client
    .create_tags()
    .resources(instance_id)
    .tags(
        Tag::builder()
            .key("Name")
            .value("From SDK Examples")
            .build(),
    )
    .send()
    .await;

match response {
    Ok(_) => tracing::info!("Created {instance_id} and applied tags."),
    Err(err) => {
        tracing::info!("Error applying tags to {instance_id}: {err:?}");
        return Err(err.into());
    }
}

```

```

    }

    tracing::info!("Instance is created.");

    Ok(instance_id.to_string())
}

```

- Untuk API detailnya, lihat [CreateTags AWS SDK untuk API referensi Rust](#).

## DeleteKeyPair

Contoh kode berikut menunjukkan cara menggunakan `DeleteKeyPair`.

### SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

Pembungkus di sekitar `delete_key` yang juga menghapus kunci pribadi pendukung. PEM

```

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(())
}
```

- Untuk API detailnya, lihat [DeleteKeyPair AWSSDK untuk API referensi Rust](#).

## DeleteSecurityGroup

Contoh kode berikut menunjukkan cara menggunakan `DeleteSecurityGroup`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [DeleteSecurityGroup AWSSDK untuk API referensi Rust](#).

## DeleteSnapshot

Contoh kode berikut menunjukkan cara menggunakan `DeleteSnapshot`.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

    println!("Deleted");

    Ok(())
}
```

- Untuk API detailnya, lihat [DeleteSnapshot AWSSDKuntuk API referensi Rust](#).

## DescribeImages

Contoh kode berikut menunjukkan cara menggunakanDescribeImages.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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)
    }
}

```

Menggunakan fungsi `list_images` dengan SSM to limit berdasarkan lingkungan Anda. Untuk detail lebih lanjut tentang SSM, lihat <https://docs.aws.amazon.com/systems-manager/latest/userguide/example-GetParameters-ssm-section.html>.

```

async fn find_image(&mut self) -> Result<ScenarioImage, EC2Error> {
    let params: Vec<Parameter> = self
        .ssm
        .list_path("/aws/service/ami-amazon-linux-latest")
        .await
        .map_err(|e| e.add_message("Could not find parameters for available
images"))?
        .into_iter()
        .filter(|param| param.name().is_some_and(|name| name.contains("amzn2")))
        .collect();
    let amzn2_images: Vec<ScenarioImage> = self
        .ec2
        .list_images(params)
        .await
        .map_err(|e| e.add_message("Could not find images"))?
        .into_iter()
        .map(ScenarioImage::from)
        .collect();
    println!("We will now create an instance from an Amazon Linux 2 AMI");
    let ami = self.util.select_scenario_image(amzn2_images)?;
    Ok(ami)
}

```

- Untuk API detailnya, lihat [DescribeImages AWS SDK](#) untuk API referensi Rust.



## DescribeInstanceStatus

Contoh kode berikut menunjukkan cara menggunakan `DescribeInstanceStatus`.

### SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [DescribeInstanceStatus AWS SDK untuk API referensi Rust](#).

## DescribeInstanceTypes

Contoh kode berikut menunjukkan cara menggunakan `DescribeInstanceTypes`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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())
}

```

- Untuk API detailnya, lihat [DescribeInstanceTypes AWSSDK](#) untuk API referensi Rust.

## DescribeInstances

Contoh kode berikut menunjukkan cara menggunakan `DescribeInstances`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

Ambil detail untuk sebuah EC2 Instance.

```

pub async fn describe_instance(&self, instance_id: &str) -> Result<Instance,
EC2Error> {
    let response = self
        .client
        .describe_instances()
        .instance_ids(instance_id)
        .send()
        .await?;

    let instance = response
        .reservations()
        .first()
        .ok_or_else(|| EC2Error::new(format!("No instance reservations for
{instance_id}")))?

```

```

        .instances()
        .first()
        .ok_or_else(|| {
            EC2Error::new(format!("No instances in reservation for
{instance_id}"))
        })?;

        Ok(instance.clone())
    }

```

Setelah membuat EC2 instance, ambil dan simpan detailnya.

```

/// 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(())
}

```

- Untuk API detailnya, lihat [DescribeInstances AWS SDK](#) untuk API referensi Rust.

## DescribeKeyPairs

Contoh kode berikut menunjukkan cara menggunakan `DescribeKeyPairs`.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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())
}
```

- Untuk API detailnya, lihat [DescribeKeyPairs AWSSDKuntuk API referensi Rust](#).

## DescribeRegions

Contoh kode berikut menunjukkan cara menggunakanDescribeRegions.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [DescribeRegions AWSSDK](#) untuk API referensi Rust.

## DescribeSecurityGroups

Contoh kode berikut menunjukkan cara menggunakan `DescribeSecurityGroups`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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");
        }
    }
}
```

```
}

```

- Untuk API detailnya, lihat [DescribeSecurityGroups AWS SDK untuk API referensi Rust](#).

## DescribeSnapshots

Contoh kode berikut menunjukkan cara menggunakan `DescribeSnapshots`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

Menampilkan status 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(())
}

```

- Untuk API detailnya, lihat [DescribeSnapshots AWSSDK](#) untuk API referensi Rust.

## DisassociateAddress

Contoh kode berikut menunjukkan cara menggunakan `DisassociateAddress`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

pub async fn disassociate_ip_address(&self, association_id: &str) -> Result<(),
EC2Error> {
    self.client
        .disassociate_address()
        .association_id(association_id)

```



```

        .send()
        .await?;
    Ok(())
}

```

- Untuk API detailnya, lihat [DisassociateAddress AWS SDK untuk API referensi Rust](#).

## RebootInstances

Contoh kode berikut menunjukkan cara menggunakan `RebootInstances`.

### SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}

```

Pelayan misalnya berada di negara berhenti dan siap, menggunakan Pelayan. API Menggunakan Waiters API membutuhkan `use aws\_sdk\_ec2: :client: :Waiters` dalam 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(())
}

```

- Untuk API detailnya, lihat [RebootInstances AWSSDK](#) untuk API referensi Rust.

## ReleaseAddress

Contoh kode berikut menunjukkan cara menggunakan `ReleaseAddress`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

pub async fn deallocate_ip_address(&self, allocation_id: &str) -> Result<(),
EC2Error> {
    self.client
        .release_address()
        .allocation_id(allocation_id)
        .send()
        .await?;
    Ok(())
}

```

- Untuk API detailnya, lihat [ReleaseAddress AWSSDK](#) untuk API referensi Rust.

## RunInstances

Contoh kode berikut menunjukkan cara menggunakan `RunInstances`.

## SDKuntuk Rust

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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())
}
```

- Untuk API detailnya, lihat [RunInstances AWSSDK](#) untuk API referensi Rust.

## StartInstances

Contoh kode berikut menunjukkan cara menggunakan `StartInstances`.

### SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode AWS](#).

Mulai EC2 Instance dengan ID Instance.

```
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(())
}
```

Tunggu instance berada dalam status siap dan status ok, menggunakan PelayanAPI. Menggunakan Waiters API membutuhkan `use aws\_sdk\_ec2: :client: :Waiters` dalam 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(())
}
```

- Untuk API detailnya, lihat [StartInstances AWSSDK](#) untuk API referensi Rust.

## StopInstances

Contoh kode berikut menunjukkan cara menggunakan `StopInstances`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

Tunggu instance berada dalam keadaan berhenti, menggunakan `Waiters API`. Menggunakan `Waiters API` membutuhkan `use aws_sdk_ec2: :client: :Waiters`` dalam 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(())
    }

```

- Untuk API detailnya, lihat [StopInstances AWS SDK untuk API referensi Rust](#).

## TerminateInstances

Contoh kode berikut menunjukkan cara menggunakan `TerminateInstances`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(())
}

```

Tunggu instans berada dalam keadaan dihentikan, menggunakan `PelayanAPI`. Menggunakan `Waiters API` membutuhkan `use aws_sdk_ec2: :client: :Waiters` dalam 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(())
}
```

- Untuk API detailnya, lihat [TerminateInstances AWSSDK untuk API referensi Rust](#).

## ECR Contoh Amazon menggunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust dengan Amazon ECR.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)

## Tindakan

### **DescribeRepositories**

Contoh kode berikut menunjukkan cara menggunakan `DescribeRepositories`.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [DescribeRepositories AWS SDKuntuk API referensi Rust](#).

## ListImages

Contoh kode berikut menunjukkan cara menggunakanListImages.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [ListImages AWSSDK](#) untuk API referensi Rust.

## ECSContoh Amazon menggunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust dengan AmazonECS.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)

## Tindakan

### CreateCluster

Contoh kode berikut menunjukkan cara menggunakan `CreateCluster`.

SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [CreateCluster AWSSDK untuk API referensi Rust](#).

### DeleteCluster

Contoh kode berikut menunjukkan cara menggunakan `DeleteCluster`.

SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}

```

- Untuk API detailnya, lihat [DeleteCluster AWS SDK untuk API referensi Rust](#).

## DescribeClusters

Contoh kode berikut menunjukkan cara menggunakan `DescribeClusters`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}

```

- Untuk API detailnya, lihat [DescribeClusters AWS SDK](#) untuk API referensi Rust.

## EKS Contoh Amazon menggunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust dengan Amazon EKS.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)

## Tindakan

### CreateCluster

Contoh kode berikut menunjukkan cara menggunakan `CreateCluster`.

SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}

```

- Untuk API detailnya, lihat [CreateCluster AWSSDKuntuk API referensi Rust](#).

## DeleteCluster

Contoh kode berikut menunjukkan cara menggunakan `DeleteCluster`.

SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}

```

- Untuk API detailnya, lihat [DeleteCluster AWS SDK](#) untuk API referensi Rust.

## AWS Glue contoh menggunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust with AWS Glue.

Dasar-dasar adalah contoh kode yang menunjukkan kepada Anda bagaimana melakukan operasi penting dalam suatu layanan.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

Memulai

Halo AWS Glue

Contoh kode berikut menunjukkan cara untuk mulai menggunakan AWS Glue.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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)),
    }
}
```



- Untuk API detailnya, lihat [ListJobs AWS SDK](#) untuk API referensi Rust.

## Topik

- [Hal-hal mendasar](#)
- [Tindakan](#)

## Hal-hal mendasar

### Pelajari dasar-dasarnya

Contoh kode berikut ini menunjukkan cara:

- Buat crawler yang merayapi bucket Amazon S3 publik dan membuat database CSV metadata yang diformat.
- Daftar informasi tentang database dan tabel di Anda AWS Glue Data Catalog.
- Buat pekerjaan untuk mengekstrak CSV data dari bucket S3, mengubah data, dan memuat output yang JSON diformat menjadi bucket S3 lain.
- Buat daftar informasi tentang menjalankan pekerjaan, melihat data yang diubah, dan membersihkan sumber daya.

Untuk informasi selengkapnya, lihat [Tutorial: Memulai AWS Glue Studio](#).

### SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode AWS](#).

Membuat dan menjalankan crawler yang merayapi bucket Amazon Simple Storage Service (Amazon S3) publik dan menghasilkan database metadata yang menjelaskan data berformat yang ditemukannya. 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)),
        }
    }
}??;

```

Daftar informasi tentang database dan tabel di Anda 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();

```

Buat dan jalankan job yang mengekstrak CSV data dari bucket Amazon S3 sumber, mengubahnya dengan menghapus dan mengganti nama bidang, dan memuat output yang JSON diformat ke bucket Amazon S3 lainnya.

```

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();

```

Hapus semua sumber daya yang dibuat oleh 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)?;
```

- Untuk API detailnya, lihat topik berikut AWS SDK untuk API referensi Rust.

- [CreateCrawler](#)
- [CreateJob](#)
- [DeleteCrawler](#)
- [DeleteDatabase](#)
- [DeleteJob](#)
- [DeleteTable](#)
- [GetCrawler](#)
- [GetDatabase](#)
- [GetDatabases](#)
- [GetJob](#)
- [GetJobRun](#)
- [GetJobRuns](#)
- [GetTables](#)
- [ListJobs](#)
- [StartCrawler](#)
- [StartJobRun](#)

## Tindakan

### CreateCrawler

Contoh kode berikut menunjukkan cara menggunakan `CreateCrawler`.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(()),
}??;
```

- Untuk API detailnya, lihat [CreateCrawler AWSSDKuntuk API referensi Rust](#).

## CreateJob

Contoh kode berikut menunjukkan cara menggunakan `CreateJob`.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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())
})?;
```

- Untuk API detailnya, lihat [CreateJob AWSSDKuntuk API referensi Rust](#).

## DeleteCrawler

Contoh kode berikut menunjukkan cara menggunakanDeleteCrawler.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

- Untuk API detailnya, lihat [DeleteCrawler AWSSDKuntuk API referensi Rust](#).

## DeleteDatabase

Contoh kode berikut menunjukkan cara menggunakanDeleteDatabase.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

- Untuk API detailnya, lihat [DeleteDatabase AWSSDKuntuk API referensi Rust](#).



## DeleteJob

Contoh kode berikut menunjukkan cara menggunakan `DeleteJob`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

- Untuk API detailnya, lihat [DeleteJob AWS SDK untuk API referensi Rust](#).

## DeleteTable

Contoh kode berikut menunjukkan cara menggunakan `DeleteTable`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

```
}
```

- Untuk API detailnya, lihat [DeleteTable AWSSDK](#) untuk API referensi Rust.

## GetCrawler

Contoh kode berikut menunjukkan cara menggunakan `GetCrawler`.

### SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode AWS](#).

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

- Untuk API detailnya, lihat [GetCrawler AWS SDK](#) untuk API referensi Rust.

## GetDatabase

Contoh kode berikut menunjukkan cara menggunakan `GetDatabase`.

### SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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()))?;

```

- Untuk API detailnya, lihat [GetDatabase AWSSDKuntuk API referensi Rust](#).

## GetJobRun

Contoh kode berikut menunjukkan cara menggunakan `GetJobRun`.

SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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();
    }

```

- Untuk API detailnya, lihat [GetJobRun AWSSDK](#) untuk API referensi Rust.

## GetTables

Contoh kode berikut menunjukkan cara menggunakan `GetTables`.

### SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

let tables = glue
    .get_tables()
    .database_name(self.database())
    .send()
    .await
    .map_err(GlueMvpError::from_glue_sdk)?;

let tables = tables.table_list();

```

- Untuk API detailnya, lihat [GetTables AWSSDK](#) untuk API referensi Rust.

## ListJobs

Contoh kode berikut menunjukkan cara menggunakan `ListJobs`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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)),
    }
}
```

- Untuk API detailnya, lihat [ListJobs AWSSDK untuk API referensi Rust](#).

## StartCrawler

Contoh kode berikut menunjukkan cara menggunakan `StartCrawler`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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)),
        }
    }
}
}??;

```

- Untuk API detailnya, lihat [StartCrawler AWS SDK untuk API referensi Rust](#).

## StartJobRun

Contoh kode berikut menunjukkan cara menggunakan `StartJobRun`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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();
```

- Untuk API detailnya, lihat [StartJobRun AWSSDK](#) untuk API referensi Rust.

## IAM contoh menggunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust with IAM.

Dasar-dasar adalah contoh kode yang menunjukkan kepada Anda bagaimana melakukan operasi penting dalam suatu layanan.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

Memulai

Halo IAM

Contoh kode berikut menunjukkan cara untuk mulai menggunakan IAM.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

## Dari src/bin/hello .rs.

```

use aws_sdk_iam::error::SdkError;
use aws_sdk_iam::operation::list_policies::ListPoliciesError;
use clap::Parser;

const PATH_PREFIX_HELP: &str = "The path prefix for filtering the results.";

#[derive(Debug, clap::Parser)]
#[command(about)]
struct HelloScenarioArgs {
    #[arg(long, default_value="/", help=PATH_PREFIX_HELP)]
    pub path_prefix: String,
}

#[tokio::main]
async fn main() -> Result<(), SdkError<ListPoliciesError>> {
    let sdk_config = aws_config::load_from_env().await;
    let client = aws_sdk_iam::Client::new(&sdk_config);

    let args = HelloScenarioArgs::parse();

    iam_service::list_policies(client, args.path_prefix).await?;

    Ok(())
}

```

## Dari 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)
}
```

- Untuk API detailnya, lihat [ListPolicies AWSSDK](#) untuk API referensi Rust.

## Topik

- [Hal-hal mendasar](#)
- [Tindakan](#)

## Hal-hal mendasar

### Pelajari dasar-dasarnya

Contoh kode berikut menunjukkan cara membuat pengguna dan mengambil peran.

#### Warning

Untuk menghindari risiko keamanan, jangan gunakan IAM pengguna untuk otentikasi saat mengembangkan perangkat lunak yang dibuat khusus atau bekerja dengan data nyata. Sebaliknya, gunakan federasi dengan penyedia identitas seperti [AWS IAM Identity Center](#).

- Buat pengguna tanpa izin.
- Buat peran yang memberikan izin untuk mencantumkan bucket Amazon S3 untuk akun tersebut.

- Tambahkan kebijakan agar pengguna dapat mengambil peran tersebut.
- Asumsikan peran dan daftar bucket S3 menggunakan kredensyal sementara, lalu bersihkan sumber daya.

## SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat topik berikut AWS SDK untuk API referensi Rust.
  - [AttachRolePolicy](#)
  - [CreateAccessKey](#)
  - [CreatePolicy](#)
  - [CreateRole](#)
  - [CreateUser](#)
  - [DeleteAccessKey](#)
  - [DeletePolicy](#)
  - [DeleteRole](#)
  - [DeleteUser](#)
  - [DeleteUserPolicy](#)
  - [DetachRolePolicy](#)
  - [PutUserPolicy](#)

## Tindakan

### **AttachRolePolicy**

Contoh kode berikut menunjukkan cara menggunakan `AttachRolePolicy`.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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
}
```

- Untuk API detailnya, lihat [AttachRolePolicy AWSSDKuntuk API referensi Rust](#).

## AttachUserPolicy

Contoh kode berikut menunjukkan cara menggunakan `AttachUserPolicy`.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}

```

- Untuk API detailnya, lihat [AttachUserPolicy AWS SDK untuk API referensi Rust](#).

## CreateAccessKey

Contoh kode berikut menunjukkan cara menggunakan `CreateAccessKey`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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())
}

```

- Untuk API detailnya, lihat [CreateAccessKey AWSSDK](#) untuk API referensi Rust.

## CreatePolicy

Contoh kode berikut menunjukkan cara menggunakan `CreatePolicy`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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())
}

```

- Untuk API detailnya, lihat [CreatePolicy AWSSDK](#) untuk API referensi Rust.

## CreateRole

Contoh kode berikut menunjukkan cara menggunakan `CreateRole`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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())
}
```

- Untuk API detailnya, lihat [CreateRole AWS SDK untuk API referensi Rust](#).

## CreateServiceLinkedRole

Contoh kode berikut menunjukkan cara menggunakan `CreateServiceLinkedRole`.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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)
}
```

- Untuk API detailnya, lihat [CreateServiceLinkedRole AWS SDKuntuk API referensi Rust](#).

## CreateUser

Contoh kode berikut menunjukkan cara menggunakanCreateUser.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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())
}
```

- Untuk API detailnya, lihat [CreateUser AWSSDK](#) untuk API referensi Rust.

## DeleteAccessKey

Contoh kode berikut menunjukkan cara menggunakan `DeleteAccessKey`.

### SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(())
}

```

- Untuk API detailnya, lihat [DeleteAccessKey AWSSDK](#) untuk API referensi Rust.

## DeletePolicy

Contoh kode berikut menunjukkan cara menggunakan `DeletePolicy`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

```

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

```

- Untuk API detailnya, lihat [DeletePolicy AWSSDK](#) untuk API referensi Rust.

## DeleteRole

Contoh kode berikut menunjukkan cara menggunakan `DeleteRole`.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [DeleteRole AWSSDKuntuk API referensi Rust](#).

## DeleteServiceLinkedRole

Contoh kode berikut menunjukkan cara menggunakanDeleteServiceLinkedRole.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}

```

- Untuk API detailnya, lihat [DeleteServiceLinkedRole AWSSDK](#) untuk API referensi Rust.

## DeleteUser

Contoh kode berikut menunjukkan cara menggunakan `DeleteUser`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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
}
```

- Untuk API detailnya, lihat [DeleteUser AWSSDKuntuk API referensi Rust](#).

## DeleteUserPolicy

Contoh kode berikut menunjukkan cara menggunakan `DeleteUserPolicy`.

SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())  
}
```

- Untuk API detailnya, lihat [DeleteUserPolicy AWSSDK](#) untuk API referensi Rust.

## DetachRolePolicy

Contoh kode berikut menunjukkan cara menggunakan `DetachRolePolicy`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(())  
}
```

- Untuk API detailnya, lihat [DetachRolePolicy AWSSDK](#) untuk API referensi Rust.

## DetachUserPolicy

Contoh kode berikut menunjukkan cara menggunakan `DetachUserPolicy`.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [DetachUserPolicy AWSSDKuntuk API referensi Rust](#).

## GetAccountPasswordPolicy

Contoh kode berikut menunjukkan cara menggunakan `GetAccountPasswordPolicy`.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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)
}
```

- Untuk API detailnya, lihat [GetAccountPasswordPolicy AWS SDK untuk API referensi Rust](#).

## GetRole

Contoh kode berikut menunjukkan cara menggunakan `GetRole`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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)
}
```

- Untuk API detailnya, lihat [GetRole AWS SDK untuk API referensi Rust](#).

## ListAttachedRolePolicies

Contoh kode berikut menunjukkan cara menggunakan `ListAttachedRolePolicies`.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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)
}
```

- Untuk API detailnya, lihat [ListAttachedRolePolicies AWSSDKuntuk API referensi Rust](#).

## ListGroups

Contoh kode berikut menunjukkan cara menggunakanListGroups.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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)
}
```

- Untuk API detailnya, lihat [ListGroups AWS SDKuntuk API referensi Rust](#).

## ListPolicies

Contoh kode berikut menunjukkan cara menggunakan `ListPolicies`.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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)
}
```

- Untuk API detailnya, lihat [ListPolicies AWSSDK](#) untuk API referensi Rust.

## ListRolePolicies

Contoh kode berikut menunjukkan cara menggunakan `ListRolePolicies`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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)
}
```

- Untuk API detailnya, lihat [ListRolePolicies AWSSDK](#) untuk API referensi Rust.

## ListRoles

Contoh kode berikut menunjukkan cara menggunakan `ListRoles`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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)
}
```

- Untuk API detailnya, lihat [ListRoles AWSSDKuntuk API referensi Rust](#).

## ListSAMLProviders

Contoh kode berikut menunjukkan cara menggunakan `ListSAMLProviders`.

SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
pub async fn list_saml_providers(
    client: &Client,
) -> Result<ListSamlProvidersOutput, SdkError<ListSAMLProvidersError>> {
    let response = client.list_saml_providers().send().await?;

    Ok(response)
}
```

- Untuk API detailnya, lihat [ListSAMLProviders AWS SDKuntuk API referensi Rust](#).

## ListUsers

Contoh kode berikut menunjukkan cara menggunakan `ListUsers`.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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)
}
```

- Untuk API detailnya, lihat [ListUsers AWS SDKuntuk API referensi Rust](#).

## AWS IoT contoh menggunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust with AWS IoT.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

### Topik

- [Tindakan](#)

## Tindakan

### DescribeEndpoint

Contoh kode berikut menunjukkan cara menggunakan `DescribeEndpoint`.

SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [DescribeEndpoint AWSSDK untuk API referensi Rust](#).

### ListThings

Contoh kode berikut menunjukkan cara menggunakan `ListThings`.

## SDKuntuk Rust

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [ListThings AWSSDKuntuk API referensi Rust](#).

## Contoh Kinesis yang digunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust with Kinesis.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)
- [Contoh nirserver](#)

## Tindakan

### CreateStream

Contoh kode berikut menunjukkan cara menggunakan CreateStream.

SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [CreateStream AWS SDK untuk API referensi Rust](#).

## DeleteStream

Contoh kode berikut menunjukkan cara menggunakan `DeleteStream`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode AWS](#).

```
async fn remove_stream(client: &Client, stream: &str) -> Result<(), Error> {
    client.delete_stream().stream_name(stream).send().await?;

    println!("Deleted stream.");

    Ok(())
}
```

- Untuk API detailnya, lihat [DeleteStream AWSSDK untuk API referensi Rust](#).

## DescribeStream

Contoh kode berikut menunjukkan cara menggunakan `DescribeStream`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(())
}

```

- Untuk API detailnya, lihat [DescribeStream AWSSDK](#) untuk API referensi Rust.

## ListStreams

Contoh kode berikut menunjukkan cara menggunakan `ListStreams`.

### SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(())
}

```

- Untuk API detailnya, lihat [ListStreams AWS SDK untuk API referensi Rust](#).

## PutRecord

Contoh kode berikut menunjukkan cara menggunakan `PutRecord`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [PutRecord AWS SDK untuk API referensi Rust](#).



## Contoh nirserver

Memanggil fungsi Lambda dari pemicu Kinesis

Contoh kode berikut menunjukkan bagaimana menerapkan fungsi Lambda yang menerima peristiwa yang dipicu oleh menerima catatan dari aliran Kinesis. Fungsi mengambil payload Kinesis, mendekode dari Base64, dan mencatat konten rekaman.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori [contoh Nirserver](#).

Mengonsumsi acara Kinesis dengan Lambda menggunakan 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
}

```

## Melaporkan kegagalan item batch untuk fungsi Lambda dengan pemicu Kinesis

Contoh kode berikut menunjukkan cara mengimplementasikan respons batch sebagian untuk fungsi Lambda yang menerima peristiwa dari aliran Kinesis. Fungsi melaporkan kegagalan item batch dalam respons, memberi sinyal ke Lambda untuk mencoba lagi pesan tersebut nanti.

### SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori [contoh Nirserver](#).

Melaporkan kegagalan item batch Kinesis dengan Lambda menggunakan 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 contoh menggunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust with AWS KMS.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)

## Tindakan

### CreateKey

Contoh kode berikut menunjukkan cara menggunakan `CreateKey`.

SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [CreateKey AWS SDK untuk API referensi Rust](#).

### Decrypt

Contoh kode berikut menunjukkan cara menggunakan `Decrypt`.

## SDKuntuk Rust

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [Dekripsi AWS](#) SDKuntuk referensi Rust API.

## Encrypt

Contoh kode berikut menunjukkan cara menggunakan `Encrypt`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [Encrypt](#) in AWS SDK untuk API referensi Rust.

## GenerateDataKey

Contoh kode berikut menunjukkan cara menggunakan `GenerateDataKey`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [GenerateDataKey AWSSDK untuk API referensi Rust](#).

## GenerateDataKeyWithoutPlaintext

Contoh kode berikut menunjukkan cara menggunakan `GenerateDataKeyWithoutPlaintext`.



## SDKuntuk Rust

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [GenerateDataKeyWithoutPlaintext AWS SDKuntuk API referensi Rust](#).

## GenerateRandom

Contoh kode berikut menunjukkan cara menggunakanGenerateRandom.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [GenerateRandom AWSSDKuntuk API referensi Rust](#).

## ListKeys

Contoh kode berikut menunjukkan cara menggunakan `ListKeys`.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [ListKeys AWSSDKuntuk API referensi Rust](#).

## ReEncrypt

Contoh kode berikut menunjukkan cara menggunakanReEncrypt.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [ReEncrypt AWS SDK untuk API referensi Rust](#).

## Contoh Lambda menggunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust with Lambda.

Dasar-dasar adalah contoh kode yang menunjukkan kepada Anda bagaimana melakukan operasi penting dalam suatu layanan.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Skenario adalah contoh kode yang menunjukkan kepada Anda bagaimana menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan atau dikombinasikan dengan yang lain Layanan AWS.

AWS kontribusi komunitas adalah contoh yang dibuat dan dikelola oleh banyak tim AWS. Untuk memberikan umpan balik, gunakan mekanisme yang disediakan di repositori terkait.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Hal-hal mendasar](#)
- [Tindakan](#)
- [Skenario](#)
- [Contoh nirserver](#)
- [AWS kontribusi komunitas](#)

## Hal-hal mendasar

Pelajari dasar-dasarnya

Contoh kode berikut ini menunjukkan cara:

- Buat IAM peran dan fungsi Lambda, lalu unggah kode handler.

- Panggil fungsi dengan satu parameter dan dapatkan hasil.
- Perbarui kode fungsi dan konfigurasi dengan variabel lingkungan.
- Panggil fungsi dengan parameter baru dan dapatkan hasil. Tampilkan log eksekusi yang dikembalikan.
- Buat daftar fungsi untuk akun Anda, lalu bersihkan sumber daya.

Untuk informasi selengkapnya, lihat [Membuat fungsi Lambda dengan konsol](#).

## SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh selengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

Cargo.toml dengan dependensi yang digunakan dalam skenario ini.

```
[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"

```

Kumpulan utilitas yang merampingkan panggilan Lambda untuk skenario ini. File ini adalah `src/operations.rs` di peti.

```

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!("{}", lambda_name.0));
        let (bucket, own_bucket) =
            match bucket {
                Some(bucket) => (Bucket(bucket), false),

```

```

        None => (
            Bucket(std::env::var("LAMBDA_BUCKET").unwrap_or_else(|_| {
                format!("rust-lambda-example-{}", uuid::Uuid::new_v4())
            })),
            true,
        ),
    };

let s3_client = aws_sdk_s3::Client::new(&sdk_config);

if own_bucket {
    info!("Creating bucket for demo: {}", bucket.0);
    s3_client
        .create_bucket()
        .bucket(bucket.0.clone())
        .create_bucket_configuration(
            CreateBucketConfiguration::builder()

.location_constraint(aws_sdk_s3::types::BucketLocationConstraint::from(
                sdk_config.region().unwrap().as_ref(),
            ))
            .build(),
        )
        .send()
        .await
        .unwrap();
}

Self::new(
    aws_sdk_iam::Client::new(&sdk_config),
    aws_sdk_lambda::Client::new(&sdk_config),
    s3_client,
    lambda_name,
    role_name,
    bucket,
    OwnBucket(own_bucket),
)
}

/**
 * Upload function code from a path to a zip file.
 * The zip file must have an AL2 Linux-compatible binary called `bootstrap`.
 * The easiest way to create such a zip is to use `cargo lambda build --output-format Zip`.

```

```
    */
    async fn prepare_function(
        &self,
        zip_file: PathBuf,
        key: Option<String>,
    ) -> Result<FunctionCode, anyhow::Error> {
        let body = ByteStream::from_path(zip_file).await?;

        let key = key.unwrap_or_else(|| format!("{}_code", self.lambda_name));

        info!("Uploading function code to s3://{}/{}", self.bucket, key);
        let _ = self
            .s3_client
            .put_object()
            .bucket(self.bucket.clone())
            .key(key.clone())
            .body(body)
            .send()
            .await?;

        Ok(FunctionCode::builder()
            .s3_bucket(self.bucket.clone())
            .s3_key(key)
            .build())
    }

    /**
     * Create a function, uploading from a zip file.
     */
    pub async fn create_function(&self, zip_file: PathBuf) -> Result<String,
    anyhow::Error> {
        let code = self.prepare_function(zip_file, None).await?;

        let key = code.s3_key().unwrap().to_string();

        let role = self.create_role().await.map_err(|e| anyhow!(e))?;

        info!("Created iam role, waiting 15s for it to become active");
        tokio::time::sleep(Duration::from_secs(15)).await;

        info!("Creating lambda function {}", self.lambda_name);
        let _ = self
            .lambda_client
            .create_function()
```

```

        .function_name(self.lambda_name.clone())
        .code(code)
        .role(role.arn())
        .runtime(aws_sdk_lambda::types::Runtime::Provided2)
        .handler("_unused")
        .send()
        .await
        .map_err(anyhow::Error::from)?;

self.wait_for_function_ready().await?;

self.lambda_client
    .publish_version()
    .function_name(self.lambda_name.clone())
    .send()
    .await?;

Ok(key)
}

/**
 * Create an IAM execution role for the managed Lambda function.
 * If the role already exists, use that instead.
 */
async fn create_role(&self) -> Result<aws_sdk_iam::types::Role, CreateRoleError>
{
    info!("Creating execution role for function");
    let get_role = self
        .iam_client
        .get_role()
        .role_name(self.role_name.clone())
        .send()
        .await;
    if let Ok(get_role) = get_role {
        if let Some(role) = get_role.role {
            return Ok(role);
        }
    }

    let create_role = self
        .iam_client
        .create_role()
        .role_name(self.role_name.clone())
        .assume_role_policy_document(ROLE_POLICY_DOCUMENT)

```

```

        .send()
        .await;

    match create_role {
        Ok(create_role) => match create_role.role {
            Some(role) => Ok(role),
            None => Err(CreateRoleError::generic(
                ErrorMetadata::builder()
                    .message("CreateRole returned empty success")
                    .build(),
            )),
        },
        Err(err) => Err(err.into_service_error()),
    }
}

/**
 * Poll `is_function_ready` with a 1-second delay. It returns when the function
 * is ready or when there's an error checking the function's state.
 */
pub async fn wait_for_function_ready(&self) -> Result<(), anyhow::Error> {
    info!("Waiting for function");
    while !self.is_function_ready(None).await? {
        info!("Function is not ready, sleeping 1s");
        tokio::time::sleep(Duration::from_secs(1)).await;
    }
    Ok(())
}

/**
 * Check if a Lambda function is ready to be invoked.
 * A Lambda function is ready for this scenario when its state is active and its
 * LastUpdateStatus is Successful.
 * Additionally, if a sha256 is provided, the function must have that as its
 * current code hash.
 * Any missing properties or failed requests will be reported as an Err.
 */
async fn is_function_ready(
    &self,
    expected_code_sha256: Option<&str>,
) -> Result<bool, anyhow::Error> {
    match self.get_function().await {
        Ok(func) => {
            if let Some(config) = func.configuration() {

```

```

        if let Some(state) = config.state() {
            info!(?state, "Checking if function is active");
            if !matches!(state, State::Active) {
                return Ok(false);
            }
        }
    }
    match config.last_update_status() {
        Some(last_update_status) => {
            info!(?last_update_status, "Checking if function is
ready");

            match last_update_status {
                LastUpdateStatus::Successful => {
                    // continue
                }
                LastUpdateStatus::Failed |
LastUpdateStatus::InProgress => {
                    return Ok(false);
                }
                unknown => {
                    warn!(
                        status_variant = unknown.as_str(),
                        "LastUpdateStatus unknown"
                    );
                    return Err(anyhow!(
                        "Unknown LastUpdateStatus, fn config is
{config:?}"
                    ));
                }
            }
        }
        None => {
            warn!("Missing last update status");
            return Ok(false);
        }
    };
    if expected_code_sha256.is_none() {
        return Ok(true);
    }
    if let Some(code_sha256) = config.code_sha256() {
        return Ok(code_sha256 ==
expected_code_sha256.unwrap_or_default());
    }
}
}
}

```

```
        Err(e) => {
            warn!(?e, "Could not get function while waiting");
        }
    }
    Ok(false)
}

/** Get the Lambda function with this Manager's name. */
pub async fn get_function(&self) -> Result<GetFunctionOutput, anyhow::Error> {
    info!("Getting lambda function");
    self.lambda_client
        .get_function()
        .function_name(self.lambda_name.clone())
        .send()
        .await
        .map_err(anyhow::Error::from)
}

/** List all Lambda functions in the current Region. */
pub async fn list_functions(&self) -> Result<ListFunctionsOutput, anyhow::Error>
{
    info!("Listing lambda functions");
    self.lambda_client
        .list_functions()
        .send()
        .await
        .map_err(anyhow::Error::from)
}

/** Invoke the lambda function using calculator InvokeArgs. */
pub async fn invoke(&self, args: InvokeArgs) -> Result<InvokeOutput,
anyhow::Error> {
    info!(?args, "Invoking {}", self.lambda_name);
    let payload = serde_json::to_string(&args)?;
    debug!(?payload, "Sending payload");
    self.lambda_client
        .invoke()
        .function_name(self.lambda_name.clone())
        .payload(Blob::new(payload))
        .send()
        .await
        .map_err(anyhow::Error::from)
}
}
```



```
/** Given a Path to a zip file, update the function's code and wait for the
update to finish. */
pub async fn update_function_code(
    &self,
    zip_file: PathBuf,
    key: String,
) -> Result<UpdateFunctionCodeOutput, anyhow::Error> {
    let function_code = self.prepare_function(zip_file, Some(key)).await?;

    info!("Updating code for {}", self.lambda_name);
    let update = self
        .lambda_client
        .update_function_code()
        .function_name(self.lambda_name.clone())
        .s3_bucket(self.bucket.clone())
        .s3_key(function_code.s3_key().unwrap().to_string())
        .send()
        .await
        .map_err(anyhow::Error::from)?;

    self.wait_for_function_ready().await?;

    Ok(update)
}

/** Update the environment for a function. */
pub async fn update_function_configuration(
    &self,
    environment: Environment,
) -> Result<UpdateFunctionConfigurationOutput, anyhow::Error> {
    info!(
        ?environment,
        "Updating environment for {}", self.lambda_name
    );
    let updated = self
        .lambda_client
        .update_function_configuration()
        .function_name(self.lambda_name.clone())
        .environment(environment)
        .send()
        .await
        .map_err(anyhow::Error::from)?;

    self.wait_for_function_ready().await?;
```

```
        Ok(updated)
    }

    /** Delete a function and its role, and if possible or necessary, its associated
code object and bucket. */
    pub async fn delete_function(
        &self,
        location: Option<String>,
    ) -> (
        Result<DeleteFunctionOutput, anyhow::Error>,
        Result<DeleteRoleOutput, anyhow::Error>,
        Option<Result<DeleteObjectOutput, anyhow::Error>>,
    ) {
        info!("Deleting lambda function {}", self.lambda_name);
        let delete_function = self
            .lambda_client
            .delete_function()
            .function_name(self.lambda_name.clone())
            .send()
            .await
            .map_err(anyhow::Error::from);

        info!("Deleting iam role {}", self.role_name);
        let delete_role = self
            .iam_client
            .delete_role()
            .role_name(self.role_name.clone())
            .send()
            .await
            .map_err(anyhow::Error::from);

        let delete_object: Option<Result<DeleteObjectOutput, anyhow::Error>> =
            if let Some(location) = location {
                info!("Deleting object {location}");
                Some(
                    self.s3_client
                        .delete_object()
                        .bucket(self.bucket.clone())
                        .key(location)
                        .send()
                        .await
                        .map_err(anyhow::Error::from),
                )
            }
    }
```

```

        } else {
            info!(?location, "Skipping delete object");
            None
        };

        (delete_function, delete_role, delete_object)
    }

pub async fn cleanup(
    &self,
    location: Option<String>,
) -> (
    (
        Result<DeleteFunctionOutput, anyhow::Error>,
        Result<DeleteRoleOutput, anyhow::Error>,
        Option<Result<DeleteObjectOutput, anyhow::Error>>,
    ),
    Option<Result<DeleteBucketOutput, anyhow::Error>>,
) {
    let delete_function = self.delete_function(location).await;

    let delete_bucket = if self.own_bucket {
        info!("Deleting bucket {}", self.bucket);
        if delete_function.2.is_none() ||
delete_function.2.as_ref().unwrap().is_ok() {
            Some(
                self.s3_client
                    .delete_bucket()
                    .bucket(self.bucket.clone())
                    .send()
                    .await
                    .map_err(anyhow::Error::from),
            )
        } else {
            None
        }
    } else {
        info!("No bucket to clean up");
        None
    };

    (delete_function, delete_bucket)
}
}

```

```

/**
 * Testing occurs primarily as an integration test running the `scenario` bin
 * successfully.
 * Each action relies deeply on the internal workings and state of Amazon Simple
 * Storage Service (Amazon S3), Lambda, and IAM working together.
 * It is therefore infeasible to mock the clients to test the individual actions.
 */
#[cfg(test)]
mod test {
    use super::{InvokeArgs, Operation};
    use serde_json::json;

    /** Make sure that the JSON output of serializing InvokeArgs is what's expected
    by the calculator. */
    #[test]
    fn test_serialize() {
        assert_eq!(json!(InvokeArgs::Increment(5)), 5);
        assert_eq!(
            json!(InvokeArgs::Arithmetic(Operation::Plus, 5, 7)).to_string(),
            r#"{"op":"plus","i":5,"j":7}"#.to_string(),
        );
    }
}

```

Biner untuk menjalankan skenario dari depan ke ujung, menggunakan flag baris perintah untuk mengontrol beberapa perilaku. File ini `src/bin/scenario` adalah `rs` di peti.

```

/*
## Service actions

Service actions wrap the SDK call, taking a client and any specific parameters
necessary for the call.

* CreateFunction
* GetFunction
* ListFunctions
* Invoke
* UpdateFunctionCode
* UpdateFunctionConfiguration
* DeleteFunction

```

## ## Scenario

A scenario runs at a command prompt and prints output to the user on the result of each service action. A scenario can run in one of two ways: straight through, printing out progress as it goes, or as an interactive question/answer script.

## ## Getting started with functions

Use an SDK to manage AWS Lambda functions: create a function, invoke it, update its code, invoke it again, view its output and logs, and delete it.

This scenario uses two Lambda handlers:

Note: Handlers don't use AWS SDK API calls.

The increment handler is straightforward:

1. It accepts a number, increments it, and returns the new value.
2. It performs simple logging of the result.

The arithmetic handler is more complex:

1. It accepts a set of actions ['plus', 'minus', 'times', 'divided-by'] and two numbers, and returns the result of the calculation.
2. It uses an environment variable to control log level (such as DEBUG, INFO, WARNING, ERROR).

It logs a few things at different levels, such as:

- \* DEBUG: Full event data.
- \* INFO: The calculation result.
- \* WARN~ING~: When a divide by zero error occurs.
- \* This will be the typical `RUST\_LOG` variable.

The steps of the scenario are:

1. Create an AWS Identity and Access Management (IAM) role that meets the following requirements:
  - \* Has an `assume_role` policy that grants 'lambda.amazonaws.com' the 'sts:AssumeRole' action.
  - \* Attaches the 'arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole' managed role.
  - \* You must wait for ~10 seconds after the role is created before you can use it!
2. Create a function (CreateFunction) for the increment handler by packaging it as a zip and doing one of the following:
  - \* Adding it with CreateFunction Code.ZipFile.

```

* --or--
* Uploading it to Amazon Simple Storage Service (Amazon S3) and adding it with
CreateFunction Code.S3Bucket/S3Key.
* _Note: Zipping the file does not have to be done in code._
* If you have a waiter, use it to wait until the function is active. Otherwise,
call GetFunction until State is Active.
3. Invoke the function with a number and print the result.
4. Update the function (UpdateFunctionCode) to the arithmetic handler by packaging
it as a zip and doing one of the following:
* Adding it with UpdateFunctionCode ZipFile.
* --or--
* Uploading it to Amazon S3 and adding it with UpdateFunctionCode S3Bucket/
S3Key.
5. Call GetFunction until Configuration.LastUpdateStatus is 'Successful' (or
'Failed').
6. Update the environment variable by calling UpdateFunctionConfiguration and pass
it a log level, such as:
* Environment={'Variables': {'RUST_LOG': 'TRACE'}}
7. Invoke the function with an action from the list and a couple of values. Include
LogType='Tail' to get logs in the result. Print the result of the calculation and
the log.
8. [Optional] Invoke the function to provoke a divide-by-zero error and show the log
result.
9. List all functions for the account, using pagination (ListFunctions).
10. Delete the function (DeleteFunction).
11. Delete the role.

```

Each step should use the function created in Service Actions to abstract calling the SDK.

```

*/

use aws_sdk_lambda::{operation::invoke::InvokeOutput, types::Environment};
use clap::Parser;
use std::{collections::HashMap, path::PathBuf};
use tracing::{debug, info, warn};
use tracing_subscriber::EnvFilter;

use lambda_code_examples::actions::{
    InvokeArgs::{Arithmetic, Increment},
    LambdaManager, Operation,
};

#[derive(Debug, Parser)]
pub struct Opt {

```

```
/// The AWS Region.
#[structopt(short, long)]
pub region: Option<String>,

// The bucket to use for the FunctionCode.
#[structopt(short, long)]
pub bucket: Option<String>,

// The name of the Lambda function.
#[structopt(short, long)]
pub lambda_name: Option<String>,

// The number to increment.
#[structopt(short, long, default_value = "12")]
pub inc: i32,

// The left operand.
#[structopt(long, default_value = "19")]
pub num_a: i32,

// The right operand.
#[structopt(long, default_value = "23")]
pub num_b: i32,

// The arithmetic operation.
#[structopt(short, long, default_value = "plus")]
pub operation: Operation,

#[structopt(long)]
pub cleanup: Option<bool>,

#[structopt(long)]
pub no_cleanup: Option<bool>,
}

fn code_path(lambda: &str) -> PathBuf {
    PathBuf::from(format!("../target/lambda/{lambda}/bootstrap.zip"))
}

fn log_invoke_output(invoke: &InvokeOutput, message: &str) {
    if let Some(payload) = invoke.payload().cloned() {
        let payload = String::from_utf8(payload.into_inner());
        info!(?payload, message);
    } else {
```

```
        info!("Could not extract payload")
    }
    if let Some(logs) = invoke.log_result() {
        debug!(?logs, "Invoked function logs")
    } else {
        debug!("Invoked function had no logs")
    }
}

async fn main_block(
    opt: &Opt,
    manager: &LambdaManager,
    code_location: String,
) -> Result<(), anyhow::Error> {
    let invoke = manager.invoke(Increment(opt.inc)).await?;
    log_invoke_output(&invoke, "Invoked function configured as increment");

    let update_code = manager
        .update_function_code(code_path("arithmetic"), code_location.clone())
        .await?;

    let code_sha256 = update_code.code_sha256().unwrap_or("Unknown SHA");
    info!(?code_sha256, "Updated function code with arithmetic.zip");

    let arithmetic_args = Arithmetic(opt.operation, opt.num_a, opt.num_b);
    let invoke = manager.invoke(arithmetic_args).await?;
    log_invoke_output(&invoke, "Invoked function configured as arithmetic");

    let update = manager
        .update_function_configuration(
            Environment::builder()
                .set_variables(Some(HashMap::from([
                    "RUST_LOG".to_string(),
                    "trace".to_string(),
                ])))
                .build(),
        )
        .await?;
    let updated_environment = update.environment();
    info!(?updated_environment, "Updated function configuration");

    let invoke = manager
        .invoke(Arithmetic(opt.operation, opt.num_a, opt.num_b))
        .await?;
```



```
log_invoke_output(
    &invoke,
    "Invoked function configured as arithmetic with increased logging",
);

let invoke = manager
    .invoke(Arithmetic(Operation::DividedBy, opt.num_a, 0))
    .await?;
log_invoke_output(
    &invoke,
    "Invoked function configured as arithmetic with divide by zero",
);

Ok::<(), anyhow::Error><()>
}

#[tokio::main]
async fn main() {
    tracing_subscriber::fmt()
        .without_time()
        .with_file(true)
        .with_line_number(true)
        .with_env_filter(EnvFilter::from_default_env())
        .init();

    let opt = Opt::parse();
    let manager = LambdaManager::load_from_env(opt.lambda_name.clone(),
    opt.bucket.clone()).await;

    let key = match manager.create_function(code_path("increment")).await {
        Ok(init) => {
            info!(?init, "Created function, initially with increment.zip");
            let run_block = main_block(&opt, &manager, init.clone()).await;
            info!(?run_block, "Finished running example, cleaning up");
            Some(init)
        }
        Err(err) => {
            warn!(?err, "Error happened when initializing function");
            None
        }
    };

    if Some(false) == opt.cleanup || Some(true) == opt.no_cleanup {
        info!("Skipping cleanup")
    }
}
```

```
    } else {  
        let delete = manager.cleanup(key).await;  
        info!(?delete, "Deleted function & cleaned up resources");  
    }  
}
```

- Untuk API detailnya, lihat topik berikut AWS SDK untuk API referensi Rust.
  - [CreateFunction](#)
  - [DeleteFunction](#)
  - [GetFunction](#)
  - [Memohon](#)
  - [ListFunctions](#)
  - [UpdateFunctionCode](#)
  - [UpdateFunctionConfiguration](#)

## Tindakan

### CreateFunction

Contoh kode berikut menunjukkan cara menggunakan `CreateFunction`.

#### SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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())
}

```

- Untuk API detailnya, lihat [CreateFunction AWSSDK](#) untuk API referensi Rust.

## DeleteFunction

Contoh kode berikut menunjukkan cara menggunakan `DeleteFunction`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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)
}
```

- Untuk API detailnya, lihat [DeleteFunction AWSSDK](#) untuk API referensi Rust.

## GetFunction

Contoh kode berikut menunjukkan cara menggunakan `GetFunction`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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)
}
```

- Untuk API detailnya, lihat [GetFunction AWSSDK untuk API referensi Rust](#).

## Invoke

Contoh kode berikut menunjukkan cara menggunakan `Invoke`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(invoke: &InvokeOutput, message: &str) {
    if let Some(payload) = invoke.payload().cloned() {
        let payload = String::from_utf8(payload.into_inner());
        info!(?payload, message);
    } else {
        info!("Could not extract payload")
    }
    if let Some(logs) = invoke.log_result() {
        debug!(?logs, "Invoked function logs")
    } else {
        debug!("Invoked function had no logs")
    }
}
}

```

- Untuk API detailnya, lihat [Memanggil AWS SDK untuk API referensi Rust](#).

## ListFunctions

Contoh kode berikut menunjukkan cara menggunakan `ListFunctions`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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)
}

```

- Untuk API detailnya, lihat [ListFunctions AWS SDK](#) untuk API referensi Rust.

## UpdateFunctionCode

Contoh kode berikut menunjukkan cara menggunakan `UpdateFunctionCode`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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())
}

```

- Untuk API detailnya, lihat [UpdateFunctionCode AWSSDK](#) untuk API referensi Rust.

## UpdateFunctionConfiguration

Contoh kode berikut menunjukkan cara menggunakan `UpdateFunctionConfiguration`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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)
}
```

- Untuk API detailnya, lihat [UpdateFunctionConfiguration AWSSDK untuk API referensi Rust](#).

## Skenario

Membuat aplikasi nirserver untuk mengelola foto

Contoh kode berikut menunjukkan cara membuat aplikasi tanpa server yang memungkinkan pengguna mengelola foto menggunakan label.

SDK untuk Rust

Menunjukkan cara mengembangkan aplikasi manajemen aset foto yang mendeteksi label dalam gambar menggunakan Amazon Rekognition dan menyimpannya untuk pengambilan nanti.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Untuk mendalami tentang asal usul contoh ini, lihat postingan di [Komunitas AWS](#).

Layanan yang digunakan dalam contoh ini

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

## Contoh nirserver

Menghubungkan ke RDS database Amazon dalam fungsi Lambda

Contoh kode berikut menunjukkan bagaimana menerapkan fungsi Lambda yang menghubungkan ke database. RDS Fungsi membuat permintaan database sederhana dan mengembalikan hasilnya.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori [contoh Nirserver](#).

## Menghubungkan ke RDS database Amazon dalam fungsi Lambda menggunakan 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}")
}))
}
```

## Memanggil fungsi Lambda dari pemicu Kinesis

Contoh kode berikut menunjukkan bagaimana menerapkan fungsi Lambda yang menerima peristiwa yang dipicu oleh menerima catatan dari aliran Kinesis. Fungsi mengambil payload Kinesis, mendekode dari Base64, dan mencatat konten rekaman.

### SDKuntuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori [contoh Nirserver](#).

## Mengkonsumsi acara Kinesis dengan Lambda menggunakan 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
}

```

## Memanggil fungsi Lambda dari pemicu DynamoDB

Contoh kode berikut menunjukkan bagaimana menerapkan fungsi Lambda yang menerima peristiwa yang dipicu oleh menerima catatan dari aliran DynamoDB. Fungsi mengambil payload DynamoDB dan mencatat isi catatan.

### SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori [contoh Nirserver](#).

## Mengonsumsi acara DynamoDB dengan Lambda menggunakan 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(())
}
```

## Memanggil fungsi Lambda dari pemicu Amazon DocumentDB

Contoh kode berikut menunjukkan cara menerapkan fungsi Lambda yang menerima peristiwa yang dipicu dengan menerima catatan dari aliran perubahan DocumentDB. Fungsi mengambil payload DocumentDB dan mencatat isi catatan.

### SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori [contoh Nirserver](#).

Mengonsumsi acara Amazon DocumentDB dengan Lambda menggunakan Rust.

```
use lambda_runtime::{service_fn, tracing, Error, LambdaEvent};
use aws_lambda_events::{
    event::documentdb::{DocumentDbEvent, DocumentDbInnerEvent},
};

// Built with the following dependencies:
//lambda_runtime = "0.11.1"
//serde_json = "1.0"
//tokio = { version = "1", features = ["macros"] }
//tracing = { version = "0.1", features = ["log"] }
//tracing-subscriber = { version = "0.3", default-features = false, features =
    ["fmt"] }
//aws_lambda_events = "0.15.0"

async fn function_handler(event: LambdaEvent<DocumentDbEvent>) ->Result<(), Error> {

    tracing::info!("Event Source ARN: {:?}", event.payload.event_source_arn);
    tracing::info!("Event Source: {:?}", event.payload.event_source);

    let records = &event.payload.events;

    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }
}
```

```
    }

    for record in records{
        log_document_db_event(record);
    }

    tracing::info!("Document db records processed");

    // Prepare the response
    Ok(())
}

fn log_document_db_event(record: &DocumentDbInnerEvent)-> Result<(), Error>{
    tracing::info!("Change Event: {:?}", record.event);

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();

    let func = service_fn(function_handler);
    lambda_runtime::run(func).await?;
    Ok(())
}
```

## Menginvokasi fungsi Lambda dari pemicu Amazon S3

Contoh kode berikut menunjukkan cara mengimplementasikan fungsi Lambda yang menerima peristiwa yang dipicu dengan mengunggah objek ke bucket S3. Fungsi ini mengambil nama bucket S3 dan kunci objek dari parameter event dan memanggil Amazon API S3 untuk mengambil dan mencatat jenis konten objek.

## SDK untuk Rust

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori [contoh Nirserver](#).

Menggunakan peristiwa S3 dengan Lambda menggunakan 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(())
}

```

## Memanggil fungsi Lambda dari pemicu Amazon SNS

Contoh kode berikut menunjukkan cara menerapkan fungsi Lambda yang menerima peristiwa yang dipicu dengan menerima pesan dari suatu SNS topik. Fungsi mengambil pesan dari parameter peristiwa dan mencatat konten setiap pesan.

### SDKuntuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori [contoh Nirserver](#).

## Mengkonsumsi SNS acara dengan Lambda menggunakan 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
}
```

```
}
```

## Memanggil fungsi Lambda dari pemicu Amazon SQS

Contoh kode berikut menunjukkan cara menerapkan fungsi Lambda yang menerima peristiwa yang dipicu oleh menerima pesan dari antrianSQS. Fungsi mengambil pesan dari parameter peristiwa dan mencatat konten setiap pesan.

### SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan [contoh lengkapnya](#) dan pelajari cara mengatur dan menjalankannya di repositori [contoh Nirserver](#).

## Mengonsumsi SQS acara dengan Lambda menggunakan 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
}

```

## Melaporkan kegagalan item batch untuk fungsi Lambda dengan pemicu Kinesis

Contoh kode berikut menunjukkan cara mengimplementasikan respons batch sebagian untuk fungsi Lambda yang menerima peristiwa dari aliran Kinesis. Fungsi melaporkan kegagalan item batch dalam respons, memberi sinyal ke Lambda untuk mencoba lagi pesan tersebut nanti.

### SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori [contoh Nirserver](#).

## Melaporkan kegagalan item batch Kinesis dengan Lambda menggunakan 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
}
```

## Melaporkan kegagalan item batch untuk fungsi Lambda dengan pemicu DynamoDB

Contoh kode berikut menunjukkan cara mengimplementasikan respons batch sebagian untuk fungsi Lambda yang menerima peristiwa dari aliran DynamoDB. Fungsi melaporkan kegagalan item batch dalam respons, memberi sinyal ke Lambda untuk mencoba lagi pesan tersebut nanti.

### SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori [contoh Nirserver](#).

## Melaporkan kegagalan item batch DynamoDB dengan Lambda menggunakan 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
}

```

## Melaporkan kegagalan item batch untuk fungsi Lambda dengan pemicu Amazon SQS

Contoh kode berikut menunjukkan cara mengimplementasikan respons batch sebagian untuk fungsi Lambda yang menerima peristiwa dari antrian SQS. Fungsi melaporkan kegagalan item batch dalam respons, memberi sinyal ke Lambda untuk mencoba lagi pesan tersebut nanti.

### SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori [contoh Nirserver](#).

## Melaporkan kegagalan item SQS batch dengan Lambda menggunakan 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 kontribusi komunitas

### Membangun dan menguji aplikasi tanpa server

Contoh kode berikut menunjukkan cara membangun dan menguji aplikasi tanpa server menggunakan API Gateway dengan Lambda dan DynamoDB

### SDK untuk Rust

Menunjukkan cara membangun dan menguji aplikasi tanpa server yang terdiri dari API Gateway dengan Lambda dan DynamoDB menggunakan Rust. SDK

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- APIGerbang
- DynamoDB
- Lambda

## MediaLive contoh menggunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust with MediaLive.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)

## Tindakan

### ListInputs

Contoh kode berikut menunjukkan cara menggunakan ListInputs.

SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat daftar nama MediaLive masukan Anda dan ARNs di Wilayah.

```
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(())
}
```

- Untuk API detailnya, lihat [ListInputs AWSSDK](#) untuk API referensi Rust.

## MediaPackage contoh menggunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust with MediaPackage.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)

## Tindakan

### ListChannels

Contoh kode berikut menunjukkan cara menggunakan `ListChannels`.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

Daftar saluran ARNs dan deskripsi.

```
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(())
}
```

- Untuk API detailnya, lihat [ListChannels AWSSDKuntuk API referensi Rust](#).

## ListOriginEndpoints

Contoh kode berikut menunjukkan cara menggunakanListOriginEndpoints.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).



## Buat daftar deskripsi titik akhir Anda dan. 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(())
}
```

- Untuk API detailnya, lihat [ListOriginEndpoints AWS SDK untuk API referensi Rust](#).

## Contoh Amazon Polly menggunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust with Amazon Polly.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Skenario adalah contoh kode yang menunjukkan kepada Anda bagaimana menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan atau dikombinasikan dengan yang lain Layanan AWS.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

### Topik

- [Tindakan](#)
- [Skenario](#)

## Tindakan

### DescribeVoices

Contoh kode berikut menunjukkan cara menggunakan `DescribeVoices`.

SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [DescribeVoices AWS SDK untuk Rust](#) referensi API.

### ListLexicons

Contoh kode berikut menunjukkan cara menggunakan `ListLexicons`.

## SDKuntuk Rust

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [ListLexicons AWSSDKuntuk API referensi Rust](#).

## PutLexicon

Contoh kode berikut menunjukkan cara menggunakan PutLexicon.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(())
}


```

- Untuk API detailnya, lihat [PutLexicon AWSSDK untuk API referensi Rust](#).

## SynthesizeSpeech

Contoh kode berikut menunjukkan cara menggunakan SynthesizeSpeech.

## SDKuntuk Rust

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [SynthesizeSpeech AWSSDKuntuk API referensi Rust](#).

## Skenario

Mengonversi teks menjadi ucapan dan kembali ke teks

Contoh kode berikut ini menunjukkan cara:

- Gunakan Amazon Polly untuk mensintesis file input teks biasa (UTF-8) ke file audio.
- Mengunggah file audio ke bucket Amazon S3.
- Menggunakan Amazon Transcribe untuk mengonversi file audio menjadi teks.
- Tampilkan teks.

### SDK untuk Rust

Gunakan Amazon Polly untuk mensintesis file input teks biasa (UTF-8) ke file audio, mengunggah file audio ke bucket Amazon S3, menggunakan Amazon Transcribe untuk mengonversi file audio tersebut menjadi teks, dan menampilkan teks.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- Amazon Polly
- Amazon S3
- Amazon Transcribe

## QLDB contoh menggunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust with QLDB.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

## Topik

- [Tindakan](#)

## Tindakan

### CreateLedger

Contoh kode berikut menunjukkan cara menggunakan `CreateLedger`.

SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [CreateLedger AWSSDK untuk API referensi Rust](#).

### ListLedgers

Contoh kode berikut menunjukkan cara menggunakan `ListLedgers`.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [ListLedgers AWSSDKuntuk API referensi Rust](#).

## RDSContoh Amazon menggunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust dengan AmazonRDS.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

### Topik

- [Contoh nirserver](#)



## Contoh nirserver

Menghubungkan ke RDS database Amazon dalam fungsi Lambda

Contoh kode berikut menunjukkan bagaimana menerapkan fungsi Lambda yang menghubungkan ke database. RDS Fungsi membuat permintaan database sederhana dan mengembalikan hasilnya.

SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori [contoh Nirserver](#).

Menghubungkan ke RDS database Amazon dalam fungsi Lambda menggunakan 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}")
    })))
}
```

# Contoh Layanan RDS Data Amazon menggunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust with Amazon RDS Data Service.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)

## Tindakan

### ExecuteStatement

Contoh kode berikut menunjukkan cara menggunakan `ExecuteStatement`.

SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [ExecuteStatement AWSSDK](#) untuk API referensi Rust.

## Contoh Amazon Rekognition menggunakan untuk Rust SDK

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust with Amazon Rekognition.

Skenario adalah contoh kode yang menunjukkan kepada Anda bagaimana menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan atau dikombinasikan dengan yang lain Layanan AWS.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Skenario](#)

## Skenario

Membuat aplikasi nirserver untuk mengelola foto

Contoh kode berikut menunjukkan cara membuat aplikasi tanpa server yang memungkinkan pengguna mengelola foto menggunakan label.

## SDK untuk Rust

Menunjukkan cara mengembangkan aplikasi manajemen aset foto yang mendeteksi label dalam gambar menggunakan Amazon Rekognition dan menyimpannya untuk pengambilan nanti.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Untuk mendalami tentang asal usul contoh ini, lihat postingan di [Komunitas AWS](#).

Layanan yang digunakan dalam contoh ini

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

### Mendeteksi wajah dalam gambar

Contoh kode berikut ini menunjukkan cara:

- Menyimpan gambar di bucket Amazon S3.
- Menggunakan Amazon Rekognition untuk mendeteksi detail wajah, seperti rentang usia, jenis kelamin, dan emosi (seperti tersenyum).
- Menampilkan detail tersebut.

## SDK untuk Rust

Menyimpan gambar di bucket Amazon S3 dengan prefiks unggahan, menggunakan Amazon Rekognition untuk mendeteksi detail wajah, seperti rentang usia, jenis kelamin, dan emosi (tersenyum, dll.), dan menampilkan detail tersebut.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- Amazon Rekognition

- Amazon S3

Simpan EXIF dan informasi gambar lainnya

Contoh kode berikut ini menunjukkan cara:

- Dapatkan EXIF informasi dariJPG,JPEG, atau PNG file.
- Mengunggah file gambar ke bucket Amazon S3.
- Menggunakan Amazon Rekognition untuk mengidentifikasi tiga atribut teratas (label) dalam file.
- Tambahkan informasi EXIF dan label ke tabel Amazon DynamoDB di Wilayah.

SDKuntuk Rust

Mendapatkan EXIF informasi dariJPG,JPEG, atau PNG file, upload file gambar ke bucket Amazon S3, gunakan Amazon Rekognition untuk mengidentifikasi tiga atribut teratas (label di Amazon Rekognition) dalam file, dan menambahkan informasi dan EXIF label ke tabel Amazon DynamoDB di Wilayah.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Amazon Rekognition
- Amazon S3

## Route 53 contoh menggunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust with Route 53.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

## Topik

- [Tindakan](#)

## Tindakan

### ListHostedZones

Contoh kode berikut menunjukkan cara menggunakan `ListHostedZones`.

#### SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(())
}
```



- Untuk API detailnya, lihat [ListHostedZones AWS SDK](#) untuk API referensi Rust.

## Contoh Amazon S3 menggunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust dengan Amazon S3.

Dasar-dasar adalah contoh kode yang menunjukkan kepada Anda bagaimana melakukan operasi penting dalam suatu layanan.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Skenario adalah contoh kode yang menunjukkan kepada Anda bagaimana menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan atau dikombinasikan dengan yang lain Layanan AWS.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

Memulai

Halo Amazon S3

Contoh kode berikut ini menunjukkan cara memulai menggunakan Amazon S3.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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 Utilitas 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)
    }
}
}

```

- Untuk API detailnya, lihat [ListBuckets AWSSDK](#) untuk API referensi Rust.

## Topik

- [Hal-hal mendasar](#)
- [Tindakan](#)
- [Skenario](#)
- [Contoh nirserver](#)



```
#[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(())
}

```

Tindakan umum yang digunakan oleh skenario.

```

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))
        }
    }
}
```

- Untuk API detailnya, lihat topik berikut AWS SDK untuk API referensi Rust.
  - [CopyObject](#)
  - [CreateBucket](#)
  - [DeleteBucket](#)
  - [DeleteObjects](#)
  - [GetObject](#)
  - [ListObjectsV2](#)
  - [PutObject](#)

## Tindakan

### CompleteMultipartUpload

Contoh kode berikut menunjukkan cara menggunakan `CompleteMultipartUpload`.

SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(),
    );
}

```

- Untuk API detailnya, lihat [CompleteMultipartUpload AWSSDK](#) untuk API referensi Rust.

## CopyObject

Contoh kode berikut menunjukkan cara menggunakan `CopyObject`.

### SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(())
}

```

- Untuk API detailnya, lihat [CopyObject AWSSDK](#) untuk API referensi Rust.

## CreateBucket

Contoh kode berikut menunjukkan cara menggunakan `CreateBucket`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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))
        }
    })
}


```

- Untuk API detailnya, lihat [CreateBucket AWSSDK](#) untuk API referensi Rust.

## CreateMultipartUpload

Contoh kode berikut menunjukkan cara menggunakan `CreateMultipartUpload`.

## SDKuntuk Rust

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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?;

```

- Untuk API detailnya, lihat [CreateMultipartUpload AWSSDK](#) untuk API referensi Rust.

## DeleteBucket

Contoh kode berikut menunjukkan cara menggunakan `DeleteBucket`.

## SDKuntuk Rust

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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))
            }
        }
    }
}
```

- Untuk API detailnya, lihat [DeleteBucket AWSSDKuntuk API referensi Rust](#).

## DeleteObject

Contoh kode berikut menunjukkan cara menggunakanDeleteObject.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [DeleteObject AWS SDKuntuk API referensi Rust](#).

## DeleteObjects

Contoh kode berikut menunjukkan cara menggunakanDeleteObjects.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(())
}

```

- Untuk API detailnya, lihat [DeleteObjects AWS SDK untuk API referensi Rust](#).

## GetBucketLocation

Contoh kode berikut menunjukkan cara menggunakan `GetBucketLocation`.

### SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(())
}

```

- Untuk API detailnya, lihat [GetBucketLocation AWSSDK untuk API referensi Rust](#).

## GetObject

Contoh kode berikut menunjukkan cara menggunakan `GetObject`.

### SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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)
}

```

- Untuk API detailnya, lihat [GetObject AWSSDK](#) untuk API referensi Rust.

## ListBuckets

Contoh kode berikut menunjukkan cara menggunakan `ListBuckets`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(())
}

```

- Untuk API detailnya, lihat [ListBuckets AWSSDK](#) untuk API referensi Rust.

## ListObjectVersions

Contoh kode berikut menunjukkan cara menggunakan `ListObjectVersions`.



## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [ListObjectVersions AWSSDKuntuk API referensi Rust](#).

## ListObjectsV2

Contoh kode berikut menunjukkan cara menggunakanListObjectsV2.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}

```

- Untuk API detailnya, lihat [ListObjectsV2 AWS SDK](#) untuk API referensi Rust.

## PutObject

Contoh kode berikut menunjukkan cara menggunakan `PutObject`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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)
}

```

- Untuk API detailnya, lihat [PutObject AWSSDK](#) untuk API referensi Rust.

## UploadPart

Contoh kode berikut menunjukkan cara menggunakan `UploadPart`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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?;
```

- Untuk API detailnya, lihat [UploadPart AWSSDK](#) untuk API referensi Rust.

## Skenario

Mengonversi teks menjadi ucapan dan kembali ke teks

Contoh kode berikut ini menunjukkan cara:

- Gunakan Amazon Polly untuk mensintesis file input teks biasa (UTF-8) ke file audio.
- Mengunggah file audio ke bucket Amazon S3.
- Menggunakan Amazon Transcribe untuk mengonversi file audio menjadi teks.
- Tampilkan teks.

### SDK untuk Rust

Gunakan Amazon Polly untuk mensintesis file input teks biasa (UTF-8) ke file audio, mengunggah file audio ke bucket Amazon S3, menggunakan Amazon Transcribe untuk mengonversi file audio tersebut menjadi teks, dan menampilkan teks.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- Amazon Polly
- Amazon S3
- Amazon Transcribe

## Buat presigned URL

Contoh kode berikut menunjukkan cara membuat presigned URL untuk Amazon S3 dan mengunggah objek.

### SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

Buat permintaan presigning ke objek 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(())
}
```

Buat permintaan presigning ke objek 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())
}
```

## Membuat aplikasi nirserver untuk mengelola foto

Contoh kode berikut menunjukkan cara membuat aplikasi tanpa server yang memungkinkan pengguna mengelola foto menggunakan label.

### SDK untuk Rust

Menunjukkan cara mengembangkan aplikasi manajemen aset foto yang mendeteksi label dalam gambar menggunakan Amazon Rekognition dan menyimpannya untuk pengambilan nanti.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Untuk mendalami tentang asal usul contoh ini, lihat postingan di [Komunitas AWS](#).

Layanan yang digunakan dalam contoh ini

- API Gerbang
- DynamoDB

- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Mendeteksi wajah dalam gambar

Contoh kode berikut ini menunjukkan cara:

- Menyimpan gambar di bucket Amazon S3.
- Menggunakan Amazon Rekognition untuk mendeteksi detail wajah, seperti rentang usia, jenis kelamin, dan emosi (seperti tersenyum).
- Menampilkan detail tersebut.

SDK untuk Rust

Menyimpan gambar di bucket Amazon S3 dengan prefiks unggahan, menggunakan Amazon Rekognition untuk mendeteksi detail wajah, seperti rentang usia, jenis kelamin, dan emosi (tersenyum, dll.), dan menampilkan detail tersebut.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini


- Amazon Rekognition
- Amazon S3

Mendapatkan objek dari bucket jika telah diubah

Contoh kode berikut menunjukkan cara membaca data dari objek di bucket S3, tetapi hanya jika bucket tersebut belum dimodifikasi sejak waktu pengambilan terakhir.



## SDKuntuk Rust

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [GetObject AWS SDK](#) untuk API referensi Rust.

## Simpan EXIF dan informasi gambar lainnya

Contoh kode berikut ini menunjukkan cara:

- Dapatkan EXIF informasi dari JPG, JPEG, atau PNG file.
- Mengunggah file gambar ke bucket Amazon S3.
- Menggunakan Amazon Rekognition untuk mengidentifikasi tiga atribut teratas (label) dalam file.
- Tambahkan informasi EXIF dan label ke tabel Amazon DynamoDB di Wilayah.

## SDK untuk Rust

Mendapatkan EXIF informasi dari JPG, JPEG, atau PNG file, upload file gambar ke bucket Amazon S3, gunakan Amazon Rekognition untuk mengidentifikasi tiga atribut teratas (label di Amazon Rekognition) dalam file, dan menambahkan informasi dan EXIF label ke tabel Amazon DynamoDB di Wilayah.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Amazon Rekognition
- Amazon S3

## Unit dan uji integrasi dengan SDK

Contoh kode berikut menunjukkan bagaimana contoh teknik praktik terbaik saat menulis unit dan tes integrasi menggunakan file AWS SDK.

### SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

Cargo.toml untuk contoh pengujian.

```
[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"
```

Contoh pengujian unit menggunakan automock dan pembungkus layanan.

```
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);
}
}

```

Contoh pengujian integrasi menggunakan 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(&[]);
}
}

```

## Mengunggah atau mengunduh file besar

Contoh kode berikut menunjukkan cara mengunggah atau mengunduh file besar ke dan dari Amazon S3.

Untuk informasi selengkapnya, lihat [Pengunggahan objek menggunakan unggahan multibagian](#).

## SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh selengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

## Contoh nirserver

Menginvokasi fungsi Lambda dari pemicu Amazon S3

Contoh kode berikut menunjukkan cara mengimplementasikan fungsi Lambda yang menerima peristiwa yang dipicu dengan mengunggah objek ke bucket S3. Fungsi ini mengambil nama bucket S3 dan kunci objek dari parameter event dan memanggil Amazon API S3 untuk mengambil dan mencatat jenis konten objek.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori [contoh Nirserver](#).

Menggunakan peristiwa S3 dengan Lambda menggunakan 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 Contoh AI menggunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust with SageMaker AI.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)

## Tindakan

### ListNotebookInstances

Contoh kode berikut menunjukkan cara menggunakan `ListNotebookInstances`.

SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [ListNotebookInstances AWSSDKAPIreferensi Rust](#).

## ListTrainingJobs

Contoh kode berikut menunjukkan cara menggunakan `ListTrainingJobs`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [ListTrainingJobs AWS SDK API referensi Rust](#).

## Contoh Secrets Manager menggunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust with Secrets Manager.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)

## Tindakan

### GetSecretValue

Contoh kode berikut menunjukkan cara menggunakan `GetSecretValue`.

SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [GetSecretValue AWSSDKAPIreferensi Rust](#).

## Amazon SES API v2 contoh menggunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust with Amazon SES API v2.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Skenario adalah contoh kode yang menunjukkan kepada Anda bagaimana menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan atau dikombinasikan dengan yang lain Layanan AWS.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)
- [Skenario](#)

## Tindakan

### CreateContact

Contoh kode berikut menunjukkan cara menggunakanCreateContact.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [CreateContact AWSSDKAPIreferensi Rust](#).

## CreateContactList

Contoh kode berikut menunjukkan cara menggunakanCreateContactList.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}

```

- Untuk API detailnya, lihat [CreateContactList AWSSDKAPIreferensi Rust](#).

## CreateEmailIdentity

Contoh kode berikut menunjukkan cara menggunakan `CreateEmailIdentity`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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) ),
    },
}

```

- Untuk API detailnya, lihat [CreateEmailIdentity AWS SDK API referensi Rust](#).

## CreateEmailTemplate

Contoh kode berikut menunjukkan cara menggunakan `CreateEmailTemplate`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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) ),
},
}

```

- Untuk API detailnya, lihat [CreateEmailTemplate AWSSDKAPIreferensi Rust](#).

## DeleteContactList

Contoh kode berikut menunjukkan cara menggunakan `DeleteContactList`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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}") ),
}


```

- Untuk API detailnya, lihat [DeleteContactList AWSSDKAPIreferensi Rust](#).

## DeleteEmailIdentity

Contoh kode berikut menunjukkan cara menggunakan `DeleteEmailIdentity`.

## SDKuntuk Rust

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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));
  }
}
```

- Untuk API detailnya, lihat [DeleteEmailIdentity AWSSDKAPIreferensi Rust](#).

**DeleteEmailTemplate**

Contoh kode berikut menunjukkan cara menggunakanDeleteEmailTemplate.

## SDKuntuk Rust

 Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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}"));
        }
    }
}

```

- Untuk API detailnya, lihat [DeleteEmailTemplate AWSSDKAPIreferensi Rust](#).

## GetEmailIdentity

Contoh kode berikut menunjukkan cara menggunakan `GetEmailIdentity`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode AWS](#).

Menentukan apakah alamat email telah diverifikasi.

```

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(())  
}
```

- Untuk API detailnya, lihat [GetEmailIdentity AWSSDKAPIreferensi Rust](#).

## ListContactLists

Contoh kode berikut menunjukkan cara menggunakan `ListContactLists`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(())  
}
```

- Untuk API detailnya, lihat [ListContactLists AWSSDKAPIreferensi Rust](#).

## ListContacts

Contoh kode berikut menunjukkan cara menggunakan `ListContacts`.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [ListContacts AWSSDKAPI](#) referensi Rust.

## SendEmail

Contoh kode berikut menunjukkan cara menggunakan `SendEmail`.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

Mengirim pesan ke semua anggota daftar kontak.

```
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(())
}

```

Mengirim pesan ke semua anggota daftar kontak menggunakan template.

```

    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)),
    }

```

- Untuk API detailnya, lihat [SendEmail AWSSDKAPIreferensi Rust](#).

## Skenario

### Alur kerja buletin

Contoh kode berikut menunjukkan cara menjalankan alur kerja buletin Amazon SES API v2.

### SDKuntuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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}"));  
}  
}
```

- Untuk API detailnya, lihat topik berikut AWS SDK untuk API referensi Rust.
  - [CreateContact](#)
  - [CreateContactList](#)
  - [CreateEmailIdentity](#)
  - [CreateEmailTemplate](#)
  - [DeleteContactList](#)
  - [DeleteEmailIdentity](#)
  - [DeleteEmailTemplate](#)
  - [ListContacts](#)
  - [SendEmail.sederhana](#)
  - [SendEmail.template](#)

## SNSSContoh Amazon menggunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust dengan AmazonSNS.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Skenario adalah contoh kode yang menunjukkan kepada Anda bagaimana menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan atau dikombinasikan dengan yang lain Layanan AWS.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)

- [Skenario](#)
- [Contoh nirserver](#)

## Tindakan

### CreateTopic

Contoh kode berikut menunjukkan cara menggunakan `CreateTopic`.

SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [CreateTopic AWSSDKAPIreferensi Rust](#).

### ListTopics

Contoh kode berikut menunjukkan cara menggunakan `ListTopics`.



## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [ListTopics AWSSDKAPIreferensi Rust](#).

## Publish

Contoh kode berikut menunjukkan cara menggunakanPublish.

## SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}

```

- Untuk API detailnya, lihat [Publikasikan AWS SDK](#) untuk API referensi Rust.

## Subscribe

Contoh kode berikut menunjukkan cara menggunakan `Subscribe`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode AWS](#).

Berlangganan alamat email ke suatu topik.

```

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(())
}
```

- Untuk API detailnya, lihat [Berlangganan AWS SDK](#) untuk API referensi Rust.

## Skenario

Membuat aplikasi nirserver untuk mengelola foto

Contoh kode berikut menunjukkan cara membuat aplikasi tanpa server yang memungkinkan pengguna mengelola foto menggunakan label.

SDK untuk Rust

Menunjukkan cara mengembangkan aplikasi manajemen aset foto yang mendeteksi label dalam gambar menggunakan Amazon Rekognition dan menyimpannya untuk pengambilan nanti.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Untuk mendalami tentang asal usul contoh ini, lihat postingan di [Komunitas AWS](#).

Layanan yang digunakan dalam contoh ini

- APIGerbang
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

## Contoh nirserver

Memanggil fungsi Lambda dari pemicu Amazon SNS

Contoh kode berikut menunjukkan cara menerapkan fungsi Lambda yang menerima peristiwa yang dipicu dengan menerima pesan dari suatu SNS topik. Fungsi mengambil pesan dari parameter peristiwa dan mencatat konten setiap pesan.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori [contoh Nirserver](#).

Mengonsumsi SNS acara dengan Lambda menggunakan 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
}
```

## SQSContoh Amazon menggunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust dengan AmazonSQS.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)
- [Contoh nirserver](#)

## Tindakan

### ListQueues

Contoh kode berikut menunjukkan cara menggunakan `ListQueues`.

SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode AWS](#).

Ambil SQS antrian Amazon pertama yang terdaftar di Wilayah.

```
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())
}
```

- Untuk API detailnya, lihat [ListQueues AWS SDK API referensi Rust](#).

## ReceiveMessage

Contoh kode berikut menunjukkan cara menggunakan `ReceiveMessage`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [ReceiveMessage AWS SDK API referensi Rust](#).

## SendMessage

Contoh kode berikut menunjukkan cara menggunakan `SendMessage`.

SDK untuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturannya dan menjalankannya di [Repositori Contoh Kode 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(())
}

```

- Untuk API detailnya, lihat [SendMessage AWSSDKAPI](#) referensi Rust.

## Contoh nirserver

### Memanggil fungsi Lambda dari pemicu Amazon SQS

Contoh kode berikut menunjukkan cara menerapkan fungsi Lambda yang menerima peristiwa yang dipicu oleh menerima pesan dari antrianSQS. Fungsi mengambil pesan dari parameter peristiwa dan mencatat konten setiap pesan.

### SDKuntuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori [contoh Nirserver](#).

Mengonsumsi SQS acara dengan Lambda menggunakan 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
}
```

## Melaporkan kegagalan item batch untuk fungsi Lambda dengan pemicu Amazon SQS

Contoh kode berikut menunjukkan cara mengimplementasikan respons batch sebagian untuk fungsi Lambda yang menerima peristiwa dari antrian SQS. Fungsi melaporkan kegagalan item batch dalam respons, memberi sinyal ke Lambda untuk mencoba lagi pesan tersebut nanti.

### SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori [contoh Nirserver](#).

## Melaporkan kegagalan item SQS batch dengan Lambda menggunakan 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 contoh menggunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust with AWS STS.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)

## Tindakan

### AssumeRole

Contoh kode berikut menunjukkan cara menggunakan `AssumeRole`.

SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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),
}
```

- Untuk API detailnya, lihat [AssumeRole AWS SDK API referensi Rust](#).

## Contoh Systems Manager menggunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust with Systems Manager.

Tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Tindakan](#)

## Tindakan

### DescribeParameters

Contoh kode berikut menunjukkan cara menggunakan `DescribeParameters`.

SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}

```

- Untuk API detailnya, lihat [DescribeParameters AWSSDKAPIreferensi Rust](#).

## GetParameter

Contoh kode berikut menunjukkan cara menggunakanGetParameter.

SDKuntuk Rust

### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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)
}

```

```
}
```

- Untuk API detailnya, lihat [GetParameter AWSSDKAPIreferensi Rust](#).

## PutParameter

Contoh kode berikut menunjukkan cara menggunakan `PutParameter`.

### SDK untuk Rust

#### Note

Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di [Repositori Contoh Kode 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(())
}
```

- Untuk API detailnya, lihat [PutParameter AWSSDKAPIreferensi Rust](#).

# Contoh Amazon Transcribe menggunakan SDK untuk Rust

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK for Rust with Amazon Transcribe.

Skenario adalah contoh kode yang menunjukkan kepada Anda bagaimana menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan atau dikombinasikan dengan yang lain Layanan AWS.

Setiap contoh menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode dalam konteks.

Topik

- [Skenario](#)

## Skenario

Mengonversi teks menjadi ucapan dan kembali ke teks

Contoh kode berikut ini menunjukkan cara:

- Gunakan Amazon Polly untuk mensintesis file input teks biasa (UTF-8) ke file audio.
- Mengunggah file audio ke bucket Amazon S3.
- Menggunakan Amazon Transcribe untuk mengonversi file audio menjadi teks.
- Tampilkan teks.

SDK untuk Rust

Gunakan Amazon Polly untuk mensintesis file input teks biasa (UTF-8) ke file audio, mengunggah file audio ke bucket Amazon S3, menggunakan Amazon Transcribe untuk mengonversi file audio tersebut menjadi teks, dan menampilkan teks.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di [GitHub](#).

Layanan yang digunakan dalam contoh ini

- Amazon Polly

- Amazon S3
- Amazon Transcribe



# Keamanan untuk AWS Produk atau Layanan ini

Keamanan cloud di Amazon Web Services (AWS) merupakan prioritas tertinggi. Sebagai seorang pelanggan AWS, Anda mendapatkan manfaat dari pusat data dan arsitektur jaringan yang dibangun untuk memenuhi persyaratan dari organisasi yang paling sensitif terhadap keamanan. Keamanan adalah tanggung jawab bersama antara Anda AWS dan Anda. [Model Tanggung Jawab Bersama](#) menggambarkan ini sebagai Keamanan dari Cloud dan Keamanan dalam Cloud.

Security of the Cloud - AWS bertanggung jawab untuk melindungi infrastruktur yang menjalankan semua layanan yang ditawarkan di AWS Cloud dan memberi Anda layanan yang dapat Anda gunakan dengan aman. Tanggung jawab keamanan kami adalah prioritas tertinggi di AWS, dan efektivitas keamanan kami secara teratur diuji dan diverifikasi oleh auditor pihak ketiga sebagai bagian dari [Program AWS Kepatuhan](#).

Keamanan di Cloud — Tanggung jawab Anda ditentukan oleh AWS layanan yang Anda gunakan, dan faktor-faktor lain termasuk sensitivitas data Anda, persyaratan organisasi Anda, serta undang-undang dan peraturan yang berlaku.

AWS Produk atau layanan ini mengikuti [model tanggung jawab bersama](#) melalui layanan Amazon Web Services (AWS) tertentu yang didukungnya. Untuk informasi keamanan AWS layanan, lihat [halaman dokumentasi keamanan AWS layanan](#) dan [AWS layanan yang berada dalam lingkup upaya AWS kepatuhan oleh program kepatuhan](#).

## Topik

- [Perlindungan data dalam AWS Produk atau Layanan ini](#)
- [Validasi Kepatuhan untuk AWS Produk atau Layanan ini](#)
- [Keamanan Infrastruktur untuk AWS Produk atau Layanan ini](#)
- [Menerapkan TLS versi minimum di AWS SDK for Rust](#)

## Perlindungan data dalam AWS Produk atau Layanan ini

[Model tanggung jawab AWS bersama model](#) berlaku untuk perlindungan data dalam AWS produk atau layanan ini. Seperti yang dijelaskan dalam model AWS ini, bertanggung jawab untuk melindungi infrastruktur global yang menjalankan semua AWS Cloud. Anda bertanggung jawab untuk mempertahankan kendali atas konten yang di-host pada infrastruktur ini. Anda juga bertanggung jawab atas tugas-tugas konfigurasi dan manajemen keamanan untuk Layanan AWS yang Anda

gunakan. Untuk informasi selengkapnya tentang privasi data, lihat [Privasi Data FAQ](#). Untuk informasi tentang perlindungan data di Eropa, lihat [Model Tanggung Jawab AWS Bersama dan](#) posting GDPR blog di Blog AWS Keamanan.

Untuk tujuan perlindungan data, kami menyarankan Anda melindungi Akun AWS kredensi dan mengatur pengguna individu dengan AWS IAM Identity Center atau AWS Identity and Access Management (IAM). Dengan cara itu, setiap pengguna hanya diberi izin yang diperlukan untuk memenuhi tanggung jawab tugasnya. Kami juga menyarankan supaya Anda mengamankan data dengan cara-cara berikut:

- Gunakan otentikasi multi-faktor (MFA) dengan setiap akun.
- Gunakan SSL/TLS untuk berkomunikasi dengan AWS sumber daya. Kami membutuhkan TLS 1.2 dan merekomendasikan TLS 1.3.
- Siapkan API dan log aktivitas pengguna dengan AWS CloudTrail. Untuk informasi tentang penggunaan CloudTrail jejak untuk menangkap AWS aktivitas, lihat [Bekerja dengan CloudTrail jejak](#) di AWS CloudTrail Panduan Pengguna.
- Gunakan solusi AWS enkripsi, bersama dengan semua kontrol keamanan default di dalamnya Layanan AWS.
- Gunakan layanan keamanan terkelola tingkat lanjut seperti Amazon Macie, yang membantu menemukan dan mengamankan data sensitif yang disimpan di Amazon S3.
- Jika Anda memerlukan FIPS 140-3 modul kriptografi yang divalidasi saat mengakses AWS melalui antarmuka baris perintah atau, gunakan titik akhir API FIPS Untuk informasi selengkapnya tentang FIPS titik akhir yang tersedia, lihat [Standar Pemrosesan Informasi Federal \(FIPS\) 140-3](#).

Kami sangat merekomendasikan agar Anda tidak pernah memasukkan informasi identifikasi yang sensitif, seperti nomor rekening pelanggan Anda, ke dalam tanda atau bidang isian bebas seperti bidang Nama. Ini termasuk ketika Anda bekerja dengan AWS produk atau layanan ini atau lainnya Layanan AWS menggunakan konsol, API, AWS CLI, atau AWS SDKs. Data apa pun yang Anda masukkan ke dalam tanda atau bidang isian bebas yang digunakan untuk nama dapat digunakan untuk log penagihan atau log diagnostik. Jika Anda memberikan URL ke server eksternal, kami sangat menyarankan agar Anda tidak menyertakan informasi kredensial dalam URL untuk memvalidasi permintaan Anda ke server tersebut.

# Validasi Kepatuhan untuk AWS Produk atau Layanan ini

Untuk mempelajari apakah an Layanan AWS berada dalam lingkup program kepatuhan tertentu, lihat [Layanan AWS di Lingkup oleh Program Kepatuhan Layanan AWS](#) dan pilih program kepatuhan yang Anda minati. Untuk informasi umum, lihat [Program AWS Kepatuhan Program AWS](#) .

Anda dapat mengunduh laporan audit pihak ketiga menggunakan AWS Artifact. Untuk informasi selengkapnya, lihat [Mengunduh Laporan di AWS Artifact](#) .

Tanggung jawab kepatuhan Anda saat menggunakan Layanan AWS ditentukan oleh sensitivitas data Anda, tujuan kepatuhan perusahaan Anda, dan hukum dan peraturan yang berlaku. AWS menyediakan sumber daya berikut untuk membantu kepatuhan:

- [Kepatuhan dan Tata Kelola Keamanan](#) – Panduan implementasi solusi ini membahas pertimbangan arsitektur serta memberikan langkah-langkah untuk menerapkan fitur keamanan dan kepatuhan.
- [Arsitektur untuk HIPAA Keamanan dan Kepatuhan di Amazon Web Services](#) — Whitepaper ini menjelaskan bagaimana perusahaan dapat menggunakan AWS untuk membuat HIPAA aplikasi yang memenuhi syarat.

## Note

Tidak semua Layanan AWS HIPAA memenuhi syarat. Untuk informasi selengkapnya, lihat [Referensi Layanan yang HIPAA Memenuhi Syarat](#).

- [AWS Sumber Daya AWS](#) — Kumpulan buku kerja dan panduan ini mungkin berlaku untuk industri dan lokasi Anda.
- [AWS Panduan Kepatuhan Pelanggan](#) - Memahami model tanggung jawab bersama melalui lensa kepatuhan. Panduan ini merangkum praktik terbaik untuk mengamankan Layanan AWS dan memetakan panduan untuk kontrol keamanan di berbagai kerangka kerja (termasuk Institut Standar dan Teknologi Nasional (NIST), Dewan Standar Keamanan Industri Kartu Pembayaran (PCI), dan Organisasi Internasional untuk Standardisasi ()). ISO
- [Mengevaluasi Sumber Daya dengan Aturan](#) dalam Panduan AWS Config Pengembang — AWS Config Layanan menilai seberapa baik konfigurasi sumber daya Anda mematuhi praktik internal, pedoman industri, dan peraturan.
- [AWS Security Hub](#)— Ini Layanan AWS memberikan pandangan komprehensif tentang keadaan keamanan Anda di dalamnya AWS. Security Hub menggunakan kontrol keamanan untuk sumber

daya AWS Anda serta untuk memeriksa kepatuhan Anda terhadap standar industri keamanan dan praktik terbaik. Untuk daftar layanan dan kontrol yang didukung, lihat [Referensi kontrol Security Hub](#).

- [Amazon GuardDuty](#) — Ini Layanan AWS mendeteksi potensi ancaman terhadap beban kerja Akun AWS, kontainer, dan data Anda dengan memantau lingkungan Anda untuk aktivitas mencurigakan dan berbahaya. GuardDuty dapat membantu Anda mengatasi berbagai persyaratan kepatuhan, seperti PCIDSS, dengan memenuhi persyaratan deteksi intrusi yang diamanatkan oleh kerangka kerja kepatuhan tertentu.
- [AWS Audit Manager](#) Ini Layanan AWS membantu Anda terus mengaudit AWS penggunaan Anda untuk menyederhanakan cara Anda mengelola risiko dan kepatuhan terhadap peraturan dan standar industri.

AWS Produk atau layanan ini mengikuti [model tanggung jawab bersama](#) melalui layanan Amazon Web Services (AWS) tertentu yang didukungnya. Untuk informasi keamanan AWS layanan, lihat [halaman dokumentasi keamanan AWS layanan](#) dan [AWS layanan yang berada dalam lingkup upaya AWS kepatuhan oleh program kepatuhan](#).

## Keamanan Infrastruktur untuk AWS Produk atau Layanan ini

AWS Produk atau layanan ini menggunakan layanan terkelola, dan karenanya dilindungi oleh keamanan jaringan AWS global. Untuk informasi tentang layanan AWS keamanan dan cara AWS melindungi infrastruktur, lihat [Keamanan AWS Cloud](#). Untuk mendesain AWS lingkungan Anda menggunakan praktik terbaik untuk keamanan infrastruktur, lihat [Perlindungan Infrastruktur dalam Kerangka Kerja](#) yang AWS Diarsiteksikan dengan Baik Pilar Keamanan.

Anda menggunakan API panggilan yang AWS dipublikasikan untuk mengakses AWS Produk atau Layanan ini melalui jaringan. Klien harus mendukung hal-hal berikut:

- Keamanan Lapisan Transportasi (TLS). Kami membutuhkan TLS 1.2 dan merekomendasikan TLS 1.3.
- Suite cipher dengan kerahasiaan maju yang sempurna (PFS) seperti (Ephemeral Diffie-Hellman) atau DHE (Elliptic Curve Ephemeral Diffie-Hellman). ECDHE Sebagian besar sistem modern seperti Java 7 dan setelahnya mendukung mode ini.

Selain itu, permintaan harus ditandatangani menggunakan access key ID dan secret access key yang terkait dengan IAM utama. Atau Anda dapat menggunakan [AWS Security Token Service](#) (AWS STS) untuk membuat kredensial keamanan sementara untuk menandatangani permintaan.

AWS Produk atau layanan ini mengikuti [model tanggung jawab bersama](#) melalui layanan Amazon Web Services (AWS) tertentu yang didukungnya. Untuk informasi keamanan AWS layanan, lihat [halaman dokumentasi keamanan AWS layanan](#) dan [AWS layanan yang berada dalam lingkup upaya AWS kepatuhan oleh program kepatuhan](#).

## Menerapkan TLS versi minimum di AWS SDK for Rust

AWS SDK for Rust Penggunaan TLS untuk meningkatkan keamanan saat berkomunikasi dengan AWS layanan. Ini SDK memberlakukan TLS versi minimum 1.2 secara default. Secara default, SDK juga menegosiasikan versi tertinggi TLS yang tersedia untuk aplikasi klien dan layanan. Misalnya, SDK mungkin bisa bernegosiasi TLS 1.3.

TLSVersi tertentu dapat diberlakukan dalam aplikasi dengan menyediakan konfigurasi manual TCP konektor yang SDK digunakan. Untuk mengilustrasikan hal ini, contoh berikut menunjukkan kepada Anda bagaimana menerapkan 1.3TLS.

### Note

Beberapa AWS layanan belum mendukung TLS 1.3, jadi menegakkan versi ini dapat memengaruhi SDK interoperabilitas. Kami merekomendasikan pengujian konfigurasi ini dengan setiap layanan sebelum penerapan produksi.

```
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(())
}
```

## Lampiran: Peti karat yang digunakan oleh SDK

Lampiran ini berisi informasi lanjutan tentang peti yang digunakan oleh AWS SDK for Rust. Ini termasuk komponen Smithy yang digunakannya, peti yang mungkin perlu Anda gunakan dalam keadaan build tertentu, dan informasi lainnya.

### Peti pandai besi

AWS SDK for Rust ini didasarkan pada [Smithy](#), seperti kebanyakan. AWS SDKs Smithy adalah bahasa yang digunakan untuk menggambarkan tipe data dan fungsi yang ditawarkan oleh SDK. Model-model ini kemudian digunakan untuk membantu membangun SDK dirinya sendiri.

[Saat melihat versi peti SDK untuk Rust dan versi dependensi Smithy-nya, mungkin akan membantu untuk mengetahui bahwa semua peti ini menggunakan penomoran versi semantik standar.](#)

Untuk informasi rinci tambahan tentang peti Smithy untuk Rust, lihat Desain Karat [Smithy](#).

### Peti digunakan dengan SDK untuk Rust

Ada sejumlah peti Smithy yang diterbitkan oleh AWS. Beberapa di antaranya relevan SDK untuk pengguna Rust, sementara yang lain adalah detail implementasi:

`aws-smithy-async`

Sertakan peti ini jika Anda tidak menggunakan Tokio untuk fungsionalitas asinkron.

`aws-smithy-runtime`

Termasuk blok bangunan yang dibutuhkan oleh semua AWS SDKs.

`aws-smithy-runtime-api`

Antarmuka yang mendasari yang digunakan oleh SDK

`aws-smithy-types`

Jenis diekspor kembali dari yang lain. AWS SDKs Gunakan ini jika Anda menggunakan beberapa SDKs.

`aws-smithy-types-convert`

Fungsi utilitas untuk bergerak masuk dan keluar `aws-smithy-types`.

## Peti lainnya

Peti berikut ada, tetapi Anda tidak perlu tahu apa-apa tentang mereka:

Peti terkait server yang SDK untuk pengguna Rust tidak perlu:

- `aws-smithy-http-server`
- `aws-smithy-http-server-python`

Peti yang berisi under-the-hood kode yang tidak perlu digunakan SDK pengguna:

- `aws-smithy-checksum-callbacks`
- `aws-smithy-eventstream`
- `aws-smithy-http`
- `aws-smithy-protocol-test`
- `aws-smithy-query`
- `aws-smithy-json`
- `aws-smithy-xml`

Peti yang tidak didukung dan akan hilang di masa depan:

- `aws-smithy-client`
- `aws-smithy-http-auth`
- `aws-smithy-http-tower`



# Riwayat dokumen

Topik ini menjelaskan perubahan penting pada Panduan AWS SDK for Rust Pengembang selama sejarahnya.

Perubahan	Deskripsi	Tanggal
<a href="#">Reorganisasi Daftar Isi</a>	Menata ulang Daftar Isi untuk membedakan konfigurasi dari penggunaan dengan lebih baik.	Juli 1, 2024
<a href="#">Ketersediaan umum dari AWS SDK for Rust</a>	Memperbarui panduan untuk menyertakan informasi keamanan baru, contoh kode baru dan yang diperbarui, detail baru tentang pengujian unit dengan contoh, dan konten baru dan terbaru lainnya untuk rilis Ketersediaan Umum yang baru SDK.	27 November 2023
<a href="#">Menegakkan versi minimum TLS</a>	Menambahkan informasi tentang cara menegakkan versi TLS di SDK.	4 Mei, 2022
<a href="#">AWS SDK for Rust rilis pratinjau pengembang</a>	<a href="#">Rilis pratinjau pengembang</a>	2 Desember 2021

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.