Panduan Developer

AWS SDK untuk .NET (V4)

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS SDK untuk .NET (V4): Panduan Developer

Copyright © 2026 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 mungkin tidak terafiliasi, terkait dengan, atau disponsori oleh Amazon.

Table of Contents

..... x
Apa itu AWS SDK untuk .NET 1
Tentang versi ini 1
Pemeliharaan dan dukungan untuk versi utama SDK 2
Kasus penggunaan umum 2
Topik tambahan di bagian ini 2
AWS Alat terkait 3
Alat untuk Windows PowerShell dan Alat untuk PowerShell Inti 3
Toolkit for VS Code 3
Toolkit for Visual Studio 3
Toolkit untuk Azure DevOps 4
Platform yang didukung 4
.NET Core 4
.NET Standar 2.0 4
.NET Kerangka 4.5 4
.NET Framework 3.5 5
Perpustakaan Kelas Portabel dan Xamarin 5
Dukungan persatuan 5
SDKs dan Referensi Alat 5
Riwayat revisi 5
Apa yang baru 6
Sumber daya tambahan 8
Memulai 10
Menginstal dan mengonfigurasi rantai alat Anda 10
Pengembangan lintas platform 10
Windows dengan Visual Studio dan .NET Core 11
Langkah berikutnya 11
Autentikasi dengan AWS 11
Login menggunakan kredenal konsol 12
Aktifkan dan konfigurasikan Pusat Identitas IAM 13
Konfigurasikan SDK untuk menggunakan IAM Identity Center. 13
Mulai sesi portal AWS akses 14
Informasi tambahan 15
Membuat aplikasi sederhana 16

Aplikasi lintas platform sederhana 16
Aplikasi berbasis Windows sederhana 22
Langkah selanjutnya 28
Melakukan konfigurasi 29
Memulai proyek baru 29
Instal AWSSDK paket dengan NuGet 31
Menggunakan NuGet dari Command prompt atau terminal 32
Menggunakan NuGet dari Visual Studio Solution Explorer 32
Menggunakan NuGet dari Package Manager Console 33
Instal AWSSDK rakitan tanpa NuGet 34
AWS Wilayah 35
Buat klien layanan dengan Wilayah tertentu 36
Tentukan Wilayah untuk semua klien layanan 37
Resolusi wilayah 38
Informasi khusus tentang Wilayah Tiongkok (Beijing) 38
Informasi khusus tentang AWS layanan baru 38
Resolusi kredensi dan profil 39
Resolusi profil 40
Menggunakan kredensil akun pengguna federasi 40
Menentukan peran atau kredensi sementara 41
Menggunakan kredensil proxy 41
Mencoba lagi dan batas waktu 42
Percobaan ulang 42
Timeout 44
Observabilitas 46
Sumber daya tambahan 47
Konfigurasikan `TelemetryProvider` 47
Metrik 49
Penyedia telemetri 51
Pengguna dan peran 52
Pengguna dan set izin 52
Peran layanan 53
Konfigurasi lanjutan 54
AWSSDK.Ekstensi. NETCore.Setup dan IConfiguration 54
Mengkonfigurasi Parameter Aplikasi Lainnya 59
Referensi File Konfigurasi untuk AWS SDK untuk .NET 67

Menggunakan kredensial warisan 79
Peringatan penting dan panduan untuk kredensional 80
Menggunakan file AWS kredensial bersama 81
Menggunakan SDK Store (khusus Windows) 84
Menggunakan SDK 88
Pemrograman asinkron 88
Paginasi 90
Di mana saya menemukan paginator? 90
Apa yang diberikan paginator kepada saya? 90
Pagination sinkron vs asinkron 91
Contoh 91
Pertimbangan tambahan untuk paginator 95
Support untuk HTTP 2 95
Pertimbangan tambahan 98
Alat tambahan 98
AWS Menyebarkan Alat 98
AWS Kerangka Pemrosesan Pesan untuk.NET 99
Integrasi dengan .NET Aspire 99
Autentikasi tingkat lanjut 100
Sign-on tunggal 100
Prasyarat 101
Menyiapkan profil SSO 101
Menghasilkan dan menggunakan token SSO 103
Sumber daya tambahan 108
Tutorial 108
Tutorial: Aplikasi .NET saja 108
Tutorial: AWS CLI dan aplikasi.NET 117
Menyebarkan ke AWS 127
Terapkan dari .NET CLI 127
Terapkan dari toolkit IDE 127
Kasus penggunaan 128
Aplikasi ASP.NET Core 128
Aplikasi Konsol .NET 129
Aplikasi Blazor WebAssembly 130
AWS Lambda proyek 130
Prasyarat 131

Perintah Lambda yang tersedia 131
Langkah-langkah untuk menyebarkan 132
AWS Layanan panggilan 133
Contoh kode terpandu 133
CloudFormation 134
Amazon Cognito 138
DynamoDB 147
Amazon EC2 176
IAM 239
Amazon S3 258
Amazon SNS 268
Amazon SQS 272
AWS Lambda 306
APIs 306
Prasyarat 306
Informasi tambahan 306
Topik 306
Anotasi Lambda 306
Pustaka dan kerangka kerja tingkat tinggi 308
Kerangka Pemrosesan Pesan 309
Integrasi AWS dengan .NET Aspire 329
Layanan dan konfigurasi lainnya 330
Contoh kode 331
Aurora 332
Memulai 332
Hal-hal mendasar 334
Tindakan 359
Auto Scaling 373
Memulai 332
Hal-hal mendasar 334
Tindakan 359
Amazon Bedrock 409
Memulai 332
Tindakan 359
Runtime Amazon Bedrock 413
Antropik Claude 413

Perintah Cohere 419
Meta Llama 423
Mistral AI 426
CloudFormation 430
Memulai 332
CloudWatch 434
Memulai 332
Hal-hal mendasar 334
Tindakan 359
CloudWatch Log 491
Tindakan 359
Skenario 495
Penyedia Identitas Amazon Cognito 517
Tindakan 359
AWS Control Tower 518
Memulai 332
Hal-hal mendasar 334
Tindakan 359
DynamoDB 557
Memulai 332
Hal-hal mendasar 334
Tindakan 359
Amazon EC2 598
Ayo Memulai 332
Amazon ECS 600
Memulai 332
AWS IoT 602
Memulai 332
Hal-hal mendasar 334
Tindakan 359
AWS IoT data 661
Tindakan 359
Amazon Redshift 664
Memulai 332
Hal-hal mendasar 334
Tindakan 359

Amazon S3 694
Memulai 332
Hal-hal mendasar 334
Tindakan 359
Skenario 495
Migrasi proyek Anda 731
Migrasi ke versi 4 731
.NET Framework 731
Tipe nilai 731
Koleksi 732
AWS Security Token Service (STS) 732
Perubahan yang berkaitan dengan `ClientConfig` 733
AWSSDK.Extensions. NETCore.Setup paket NuGet 733
`CookieSigner` dan `UrlSigner` 734
DateTime versus UTC DateTime 735
DateTime penguraian 735
`ConvertFromUnixEpochSeconds` dan `ConvertFromUnixEpochMilliseconds` 736
Pencatatan log 736
Support untuk HTTP 2 737
Sign-on tunggal 737
Perubahan khusus untuk DynamoDB 737
Perubahan khusus untuk EC2 741
Perubahan khusus untuk S3 742
Elemen pemrograman yang telah dihapus 745
Migrasi dari .NET Standard 1.3 748
Keamanan 750
Perlindungan data 751
Identity and Access Management 752
Audiens 752
Mengautentikasi dengan identitas 753
Mengelola akses menggunakan kebijakan 754
Bagaimana Layanan AWS bekerja dengan IAM 756
Memecahkan masalah AWS identitas dan akses 756
Validasi Kepatuhan 758
Ketahanan 759
Keamanan Infrastruktur 759

Menegakkan versi TLS minimum 760
.NET Core 760
.NET Framework 761
Alat AWS untuk PowerShell 762
Xamarin 763
Unity 764
Browser (untuk Blazor WebAssembly) 764
Migrasi Klien Enkripsi S3 (V1 ke V2) 764
Ikhtisar Migrasi 765
Perbarui Klien yang Ada ke Klien Transisi V1 untuk Membaca Format Baru 765
Migrasikan Klien Transisi V1 ke Klien V2 untuk Menulis Format Baru 766
Perbarui Klien V2 agar Tidak Lagi Membaca Format V1 769
Migrasi Klien Enkripsi S3 (V2 ke V4) 770
Memahami Konsep V4 770
Perbarui Klien yang Ada untuk Membaca Format V4 772
Migrasi ke Klien V4 773
Contoh Konfigurasi Tambahan 777
Pertimbangan khusus 780
Memperoleh AWSSDK majelis 780
Unduh dan ekstrak file ZIP 780
Mengakses kredensi dan profil dalam aplikasi 781
Contoh untuk kelas CredentialProfileStoreChain 782
Contoh untuk kelas SharedCredentialsFile dan AWSCredentials Pabrik 783
Dukungan persatuan 784
Dukungan Xamarin 785
Referensi API 786
Tentang versi referensi API 786
Riwayat dokumen 789

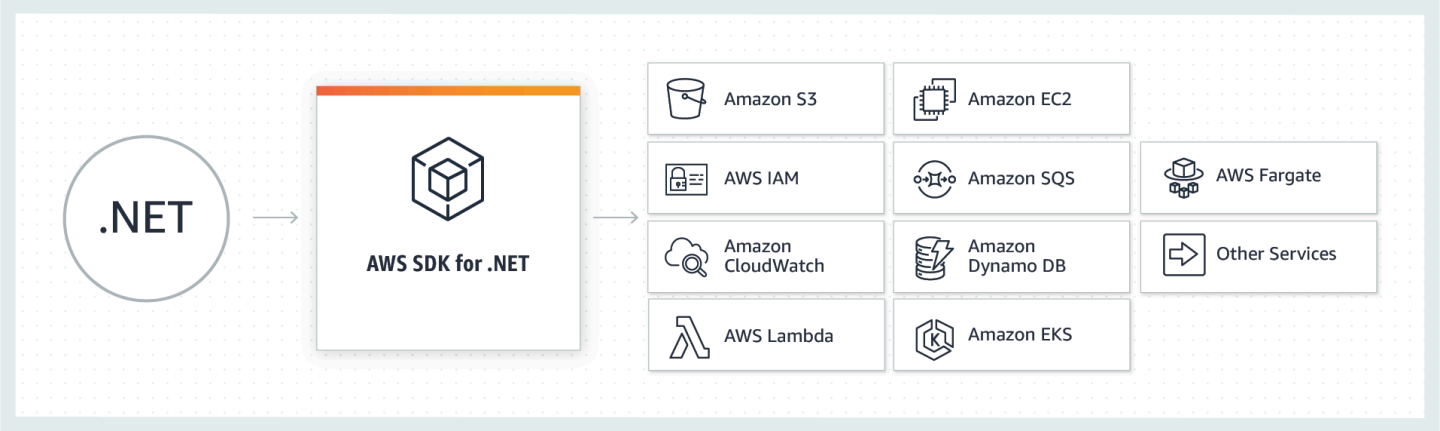
Versi 4 (V4) dari AWS SDK untuk .NET telah dirilis!

Untuk informasi tentang melanggar perubahan dan memigrasi aplikasi Anda, lihat topik migrasi.

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.

Apa itu AWS SDK untuk .NET

Ini AWS SDK untuk .NET membuatnya lebih mudah untuk membangun aplikasi.NET yang memanfaatkan AWS layanan yang hemat biaya, terukur, dan andal seperti Amazon Simple Storage Service (Amazon S3) Service (Amazon S3) dan Amazon Elastic Compute Cloud (Amazon). EC2 SDK menyederhanakan penggunaan AWS layanan dengan menyediakan seperangkat pustaka yang konsisten dan akrab bagi pengembang.NET..



(OK, mengerti! Saya siap untuk mengatur dan mengambil tur singkat.)

Tentang versi ini

> **Dokumentasi saat ini dan masa lalu**
>
> Dokumentasi ini untuk versi 4.0 dan yang lebih baru (V4) dari versi. AWS SDK untuk .NET Ini sebagian besar berpusat di sekitar.NET Core dan ASP.NET Core, tetapi juga berisi informasi tentang.NET Framework dan ASP.NET 4. x. Selain Windows dan Visual Studio, ini memberikan pertimbangan yang sama untuk pengembangan lintas platform.
> Untuk informasi tentang migrasi dari versi SDK sebelumnya, lihat. Migrasi proyek Anda
> Untuk informasi tentang SDK versi sebelumnya, V3, yang akan dihentikan di masa mendatang, lihat Panduan AWS SDK untuk .NET Pengembang (V3).
> Untuk menemukan konten usang untuk versi SDK yang lebih lama, lihat item berikut:
>
> - AWS SDK untuk .NET (versi 2, tidak digunakan lagi) Panduan Pengembang
> - Referensi API yang tidak digunakan lagi untuk AWS SDK untuk .NET

Pemeliharaan dan dukungan untuk versi utama SDK

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

- AWS SDKs dan kebijakan pemeliharaan alat
- AWS SDKs dan matriks dukungan versi alat

Kasus penggunaan umum

AWS SDK untuk .NET Ini membantu Anda mewujudkan beberapa kasus penggunaan yang menarik, termasuk yang berikut:

- Kelola pengguna dan peran dengan AWS Identity and Access Management (IAM).
- Akses Amazon Simple Storage Service (Amazon S3) untuk membuat bucket dan menyimpan objek.
- Kelola langganan HTTP Amazon Simple Notification Service (Amazon SNS) ke topik.
- Gunakan utilitas transfer S3 untuk mentransfer file ke Amazon S3 dari aplikasi Xamarin Anda.
- Gunakan Amazon Simple Queue Service (Amazon Simple Queue Service) untuk memproses pesan dan alur kerja antar komponen dalam sistem.
- Lakukan transfer Amazon S3 yang efisien dengan mengirimkan pernyataan SQL ke Amazon S3 Select.
- Buat dan luncurkan EC2 instans Amazon, serta konfigurasikan serta minta instans EC2 spot Amazon.

Topik tambahan di bagian ini

- AWS alat yang terkait dengan AWS SDK untuk .NET
- Platform yang didukung oleh AWS SDK untuk .NET
- AWS SDKs dan Panduan Referensi Alat
- Riwayat revisi
- Apa yang baru di AWS SDK untuk .NET
- Sumber daya tambahan

AWS alat yang terkait dengan AWS SDK untuk .NET

Alat untuk Windows PowerShell dan Alat untuk PowerShell Inti

Modul AWS Tools for Windows PowerShell dan AWS Tools for PowerShell Core merupakan PowerShell modul yang dibangun di atas fungsionalitas yang diekspos oleh AWS SDK untuk .NET. AWS PowerShell Alat ini memungkinkan Anda untuk membuat skrip operasi pada AWS sumber daya Anda dari PowerShell prompt. Meskipun cmdlet diimplementasikan menggunakan klien layanan dan metode dari SDK, cmdlet memberikan PowerShell pengalaman idiomatik untuk menentukan parameter dan menangani hasil.

Untuk memulai, lihat AWS Tools for Windows PowerShell.

Toolkit for VS Code

AWS Toolkit for Visual Studio CodeIni adalah plugin untuk editor Visual Studio Code (VS Code). Toolkit memudahkan Anda untuk mengembangkan, men-debug, dan menyebarkan aplikasi yang digunakan. AWS

Dengan toolkit, Anda dapat melakukan hal-hal seperti berikut:

- Buat aplikasi tanpa server yang berisi AWS Lambda fungsi, lalu gunakan aplikasi ke tumpukan. AWS CloudFormation
- Bekerja dengan EventBridge skema Amazon.
- Gunakan IntelliSense saat bekerja dengan file definisi tugas Amazon ECS.
- Visualisasikan AWS Cloud Development Kit (AWS CDK) aplikasi.

Toolkit for Visual Studio

AWS Toolkit for Visual Studio Ini adalah plugin untuk Visual Studio IDE yang memudahkan Anda untuk mengembangkan, men-debug, dan menyebarkan aplikasi.NET yang menggunakan Amazon Web Services. Toolkit for Visual Studio menyediakan template Visual Studio untuk layanan seperti Lambda dan penyihir penyebaran untuk aplikasi web dan aplikasi tanpa server. Anda dapat menggunakan AWS Explorer untuk mengelola EC2 instans Amazon, bekerja dengan tabel Amazon DynamoDB, mempublikasikan pesan ke antrian Simple Notification Service Amazon (Amazon SNS), dan banyak lagi, semuanya dalam Visual Studio.

Untuk memulai, lihat Menyiapkan AWS Toolkit for Visual Studio.

Toolkit untuk Azure DevOps

AWS Toolkit for Microsoft Azure DevOps Menambahkan tugas untuk mengaktifkan pipeline build dan release dengan mudah di Azure DevOps dan Azure DevOps Server untuk bekerja dengan layanan. AWS Anda dapat bekerja dengan Amazon S3,,, AWS CodeDeploy Lambda AWS Elastic Beanstalk, AWS CloudFormation, Amazon Simple Queue Service (Amazon SQS), dan Amazon SNS. Anda juga dapat menjalankan perintah menggunakan PowerShell modul Windows dan AWS Command Line Interface (AWS CLI).

Untuk memulai AWS Toolkit for Azure DevOps, lihat Panduan AWS Toolkit for Microsoft Azure DevOps Pengguna.

Platform yang didukung oleh AWS SDK untuk .NET

AWS SDK untuk .NET Ini menyediakan kelompok rakitan yang berbeda bagi pengembang untuk menargetkan platform yang berbeda. Namun, tidak semua fungsi SDK sama pada masing-masing platform ini. Topik ini menjelaskan perbedaan dukungan untuk setiap platform.

.NET Core

AWS SDK untuk .NET Mendukung aplikasi yang ditulis untuk.NET Core (.NET Core 3.1, .NET 5, .NET 6, dan sebagainya). AWS klien layanan hanya mendukung pola panggilan asinkron di inti .NET. Ini juga memengaruhi banyak abstraksi tingkat tinggi yang dibangun di atas klien layanan, seperti Amazon `TransferUtility` S3, yang hanya akan mendukung panggilan asinkron di lingkungan .NET Core.

.NET Standar 2.0

Variasi Non-Framework AWS SDK untuk .NET sesuai dengan .NET Standard 2.0. AWS SDK untuk .NET Ini hanya menyediakan metode asinkron untuk aplikasi yang ditulis terhadap Standar .NET.

.NET Kerangka 4.5

Versi ini dikompilasi terhadap .NET Framework 4.7.2 dan berjalan di runtime .NET 4.0. AWS SDK untuk .NET AWS klien layanan mendukung pola panggilan sinkron dan asinkron dan menggunakan kata kunci asinkron dan menunggu yang diperkenalkan di C# 5.0.

.NET Framework 3.5

Versi 4 AWS SDK untuk .NET tidak mendukung .NET Framework 3.5.

Perpustakaan Kelas Portabel dan Xamarin

AWS SDK untuk .NET Juga berisi implementasi Portable Class Library. Implementasi Portable Class Library dapat menargetkan beberapa platform, termasuk Universal Windows Platform (UWP) dan Xamarin di iOS dan Android. Lihat Mobile SDK untuk.NET dan Xamarin untuk detail selengkapnya. AWS klien layanan hanya mendukung pola panggilan asinkron.

Dukungan persatuan

Untuk informasi tentang dukungan Unity, lihatPertimbangan khusus untuk dukungan Unity.

AWS SDKs dan Panduan Referensi Alat

Panduan Referensi AWS SDKs and Tools berisi informasi yang relevan dan penting bagi banyak perangkat AWS SDKs dan toolkit dan. AWS CLI Berikut ini adalah beberapa contoh informasi yang terkandung dalam referensi:

- Informasi tentang yang dibagikan AWS`config` dan `credentials` file dan lokasinya.
- Menyiapkan AWS akun, pengguna, dan peran
- Referensi pengaturan konfigurasi dan otentikasi
- AWS Pustaka Runtime Umum (CRT)
- AWS SDKs dan kebijakan pemeliharaan alat
- AWS SDKs dan matriks dukungan versi alat

Riwayat revisi

Untuk mengetahui apa yang telah berubah dalam berbagai rilis, lihat yang berikut ini:

- Log perubahan SDK
- Apa yang baru di AWS SDK untuk .NET
- Riwayat dokumen

Apa yang baru di AWS SDK untuk .NET

Untuk informasi tingkat tinggi tentang perkembangan baru yang terkait dengan AWS SDK untuk .NET, lihat halaman produk di https://aws.amazon.com/sdk-for-net/dan log perubahan SDK.

Berikut ini adalah apa yang baru di AWS SDK untuk .NET.

17 Oktober 2025: Versi 2.0 dari Alat AWS Penerapan

Versi 2.0 dari Alat AWS Deploy untuk.NET CLI telah dirilis. Untuk informasi tentang Alat Deploy, lihatMenyebarkan aplikasi ke AWS. Untuk informasi tentang versi 2.0 dari Alat Deploy, lihat posting blog Apa yang Baru di Alat AWS Deploy untuk.NET.

21 Agustus 2025: Mendatang end-of-support untuk versi 3 AWS SDK untuk .NET

end-of-supportUntuk versi 3 (V3) dari AWS SDK untuk .NET telah diumumkan. Lihat panduan migrasi untuk memigrasi aplikasi V3 Anda dan menghindari gangguan. Untuk informasi lebih lanjut, lihat posting blog Mengumumkan end-of-support untuk AWS SDK untuk .NET v3.

2 Juli 2025: Ketersediaan umum Penyedia Cache Terdistribusi

Penyedia Cache Terdistribusi AWS .NET untuk Amazon DynamoDB umumnya tersedia! Penyedia ini mengimplementasikan antarmuka ASP.NET Core IDistributedCache, memungkinkan Anda mengintegrasikan infrastruktur DynamoDB yang dikelola sepenuhnya dan tahan lama ke dalam lapisan caching Anda dengan perubahan kode minimal. Penyedia Cache Terdistribusi tersedia melalui AWS. AspNetCore. DistributedCacheProviderpaket pada NuGet. Untuk informasi tambahan, lihat posting blog AWS .NET Penyedia Cache Terdistribusi untuk Amazon DynamoDB sekarang Tersedia Secara Umum.

> **Note**
>
> Versi rilis dari Penyedia Cache Terdistribusi hanya didukung pada V4 dari. AWS SDK untuk .NET Jika Anda mencoba versi pratinjau Penyedia Cache Terdistribusi, Anda mungkin perlu memperbarui dependensi paket dalam aplikasi Anda.

5 Mei 2025: Ketersediaan umum Kerangka Pemrosesan AWS Pesan untuk.NET

Kerangka Pemrosesan AWS Pesan untuk.NET umumnya tersedia! Framework adalah framework AWS-native yang menyederhanakan pengembangan aplikasi pemrosesan pesan.NET yang

menggunakan AWS layanan seperti Amazon Simple Queue Service (SQS), Amazon Simple Notification Service (SNS), dan Amazon. EventBridge

28 April 2025: Versi 4 dari AWS SDK untuk .NET

Versi 4 dari AWS SDK untuk .NET umumnya tersedia! Untuk informasi tentang memigrasi aplikasi Anda ke V4, lihat. Migrasi ke versi 4 Lihat juga posting blog Ketersediaan Umum AWS SDK untuk .NET V4.0 yang mengumumkan ketersediaan umum.

15 Februari 2025: Integrasi dengan .NET Aspire

Integrasi dengan.NET Aspire untuk meningkatkan loop dev dalam telah dirilis. Untuk informasi, lihat Integrasi AWS dengan .NET Aspire di AWS SDK untuk .NET.

10 Februari 2025: Rilis GA untuk observabilitas

Observabilitas adalah sejauh mana keadaan sistem saat ini dapat disimpulkan dari data yang dipancarkannya. Observabilitas telah ditambahkan ke AWS SDK untuk .NET, termasuk implementasi penyedia telemetri. Untuk informasi lebih lanjut, lihat Observabilitas di panduan ini dan posting blog Mengumumkan ketersediaan umum OpenTelemetry pustaka AWS .NET.

15 Januari 2025: Perilaku default baru untuk perlindungan integritas

Dimulai dengan versi 3.7.412.0 AWS SDK untuk .NET, SDK menyediakan perlindungan integritas default dengan menghitung checksum secara otomatis untuk unggahan. `CRC32` Untuk informasi lebih lanjut, lihat pengumuman GitHub dihttps://github.com/aws/aws-sdk-net/issues/3610. SDK juga menyediakan pengaturan global untuk perlindungan integritas data yang dapat Anda atur secara eksternal, yang dapat Anda baca di Perlindungan Integritas Data di Panduan Referensi Alat dan Alat.AWS SDKs

15 November 2024: Posting blog untuk rilis pratinjau 4 untuk versi 4

Pratinjau 4 AWS SDK untuk .NET Versi 4 dirilis pada 15 November 2024. Untuk informasi lebih lanjut tentang pratinjau ini, lihat posting blog Pratinjau 4 dari AWS SDK untuk .NET V4.

16 Agustus 2024: Posting blog untuk rilis pratinjau 1 untuk versi 4

AWS SDK untuk .NET Versi 4 dirilis sebagai pratinjau pertama pada 16 Agustus 2024. Untuk informasi lebih lanjut tentang pratinjau ini, lihat posting blog Pratinjau 1 dari AWS SDK untuk .NET V4.

Sumber daya tambahan

Layanan yang didukung

AWS SDK untuk .NET Mendukung sebagian besar produk AWS infrastruktur, dan lebih banyak layanan sering ditambahkan. Untuk daftar AWS layanan yang didukung oleh SDK, lihat file SDK README.

Halaman rumah untuk AWS SDK untuk .NET

Untuk informasi selengkapnya tentang AWS SDK untuk .NET, lihat halaman beranda SDK di https://aws.amazon.com/sdk-for-net/.

Dokumentasi referensi SDK

Dokumentasi referensi SDK memberi Anda kemampuan untuk menelusuri dan mencari di semua kode yang disertakan dengan SDK. Ini memberikan dokumentasi menyeluruh dan contoh penggunaan. Untuk informasi lebih lanjut, lihat Referensi API AWS SDK untuk .NET.

AWS re:post (sebelumnya forum AWS)

Kunjungi AWS re:post, khususnya topik untuk AWS SDK untuk .NET, untuk mengajukan pertanyaan atau memberikan umpan balik tentang. AWS Setiap halaman dokumentasi memiliki link Try AWS re:Post di bagian bawah halaman yang membawa Anda ke topik re:post terkait. AWS insinyur memantau topik dan menanggapi pertanyaan, umpan balik, dan masalah.

Jika Anda masuk ke re:Post, Anda juga dapat mengikuti topik. Untuk mengikuti topik AWS SDK untuk .NET, buka halaman Semua Topik, temukan ".NET on AWS", dan pilih tombol Ikuti.

Toolkit

- AWS Toolkit for Visual Studio: Jika Anda menggunakan Microsoft Visual Studio IDE, Anda harus memeriksa Panduan AWS Toolkit for Visual Studio Pengguna.
- AWS Toolkit for Visual Studio Code: Jika Anda menggunakan Microsoft Visual Studio IDE, Anda harus memeriksa Panduan AWS Toolkit for Visual Studio Code Pengguna.

Perpustakaan, ekstensi, dan alat yang bermanfaat

Kunjungi aws/dotnet dan aws/ aws-sdk-net repositori di GitHub situs web untuk tautan ke perpustakaan, alat, dan sumber daya yang dapat Anda gunakan untuk membantu membangun aplikasi dan layanan .NET. AWS

Berikut ini beberapa contohnya:

- AWS .NET Configuration Extension untuk Systems Manager
- AWS Ekstensi. NET Core Setup
- AWS Pencatangan.NET
- Perpustakaan Ekstensi Otentikasi Amazon Cognito
- AWS X-Ray SDK for .NET

Sumber daya lainnya

Berikut ini adalah sumber daya lain yang mungkin terbukti berguna:

- Pengembang net
- .NET Development Environment di AWS Cloud - Penerapan Referensi Mulai Cepat
- Halo, Cloud! blog
- AWS Whitepaper: Mengembangkan dan Menyebarkan Aplikasi.NET di AWS
- AWS Microservice Extractor for .NET
- Asisten Porting untuk .NET
- AWS SDKs dan Panduan Referensi Alat

Memulai dengan AWS SDK untuk .NET

Pelajari cara menginstal, mengatur, dan menggunakan AWS SDK untuk .NET. Anda harus menginstal toolchain Anda dan menyiapkan sejumlah hal penting yang dibutuhkan aplikasi Anda untuk mengakses AWS layanan. Ini termasuk:

- Akun atau peran pengguna yang sesuai
- Informasi otentikasi untuk akun pengguna tersebut atau untuk mengambil peran itu

Untuk informasi tentang cara mengkonfigurasi proyek dan bagian penting lainnya, lihatMengkonfigurasi AWS SDK untuk .NET.

Topik

- Menginstal dan mengonfigurasi toolchain Anda untuk AWS SDK untuk .NET
- Mengautentikasi dengan AWS SDK untuk .NET AWS
- Membuat aplikasi sederhana menggunakan AWS SDK untuk .NET

Menginstal dan mengonfigurasi toolchain Anda untuk AWS SDK untuk .NET

Untuk menggunakannya AWS SDK untuk .NET, Anda harus menginstal alat pengembangan tertentu.

Pengembangan lintas platform

Berikut ini diperlukan untuk pengembangan .NET lintas platform di Windows, Linux, atau macOS:

- Microsoft .NET Core SDK, versi 2.1, 3.1, atau yang lebih baru, yang mencakup antarmuka baris perintah .NET (CLI) **`dotnet`**() dan runtime .NET Core.

- Editor kode atau lingkungan pengembangan terintegrasi (IDE) yang sesuai untuk sistem operasi dan persyaratan Anda. Ini biasanya salah satu yang menyediakan beberapa dukungan untuk .NET Core.

  Contohnya termasuk Microsoft Visual Studio Code (VS Code), JetBrains Rider, dan Microsoft Visual Studio.

- (Opsional) AWS Toolkit jika tersedia untuk editor yang Anda pilih dan sistem operasi Anda.

  Contohnya termasuk AWS Toolkit for Visual Studio Code, AWS Toolkit for JetBrains, dan AWS Toolkit for Visual Studio.

Windows dengan Visual Studio dan .NET Core

Berikut ini diperlukan untuk pengembangan pada Windows dengan Visual Studio dan .NET Core:

- Microsoft Visual Studio

- Microsoft .NET Core 2.1, 3.1 atau yang lebih baru

  Ini biasanya disertakan secara default saat menginstal versi terbaru Visual Studio.

- (Opsional) AWS Toolkit for Visual Studio, yang merupakan plugin yang menyediakan antarmuka pengguna untuk mengelola AWS sumber daya dan profil lokal Anda dari Visual Studio. Untuk menginstal toolkit, lihat Menyiapkan file. AWS Toolkit for Visual Studio

  Untuk informasi selengkapnya, silakan lihat Panduan Pengguna AWS Toolkit for Visual Studio.

Langkah berikutnya

Mengautentikasi dengan AWS SDK untuk .NET AWS

Mengautentikasi dengan AWS SDK untuk .NET AWS

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

Untuk melihat berbagai metode otentikasi SDK, lihat Autentikasi dan akses di Panduan Referensi Alat AWS SDKs dan Alat.

Topik ini mengasumsikan bahwa pengguna baru sedang berkembang secara lokal, belum diberikan metode otentikasi oleh majikan mereka, dan akan digunakan AWS IAM Identity Center untuk mendapatkan kredensil sementara. Jika lingkungan Anda tidak termasuk dalam asumsi ini, beberapa

informasi dalam topik ini mungkin tidak berlaku untuk Anda, atau beberapa informasi mungkin telah diberikan kepada Anda.

Mengkonfigurasi lingkungan ini memerlukan beberapa langkah, yang dirangkum sebagai berikut:

1. Login menggunakan kredenal konsol
2. Aktifkan dan konfigurasikan Pusat Identitas IAM
3. Konfigurasikan SDK untuk menggunakan IAM Identity Center.
4. Mulai sesi portal AWS akses

Login menggunakan kredenal konsol

Anda dapat menggunakan kredenal masuk Konsol AWS Manajemen yang ada untuk akses terprogram ke layanan. AWS Setelah alur otentikasi berbasis browser, buat kredenal AWS sementara yang berfungsi di seluruh alat pengembangan lokal seperti AWS CLI, dan SDK Alat AWS untuk PowerShell for .NET v4. AWS

Dengan proses ini, Anda dapat mengautentikasi menggunakan kredensi root yang dibuat selama pengaturan akun awal, pengguna IAM, atau identitas federasi dari penyedia identitas Anda, dan AWS CLI atau Alat AWS untuk PowerShell secara otomatis mengelola kredensil sementara untuk Anda. Pendekatan ini meningkatkan keamanan dengan menghilangkan kebutuhan untuk menyimpan kredensil jangka panjang secara lokal. Ketika Anda menjalankan `Invoke-AWSLogin` cmdlet dari Alat AWS untuk PowerShell atau `aws login` perintah dari AWS CLI, Anda dapat memilih dari sesi konsol aktif Anda, atau masuk melalui alur otentikasi berbasis browser dan ini akan secara otomatis menghasilkan kredensi sementara. Token kredenal sementara akan kedaluwarsa dalam 15 menit, tetapi AWS SDK for .NET v4 secara otomatis menyegarkan token saat diperlukan selama permintaan Anda.

> **Important**
>
> Jika Anda menggunakan kredensi konsol untuk autentikasi, aplikasi Anda harus mereferensikan NuGet paket berikut agar resolusi AWS Signin dapat berfungsi:
>
> - AWSSDK.Masuk
>
> Kegagalan untuk mereferensikan paket-paket ini akan menghasilkan pengecualian runtime.

- Alat AWS untuk PowerShell login creds Panduan Pengembang.
- Login untuk pengembangan AWS lokal menggunakan kredensil konsol menggunakan CLI AWS .

`config`File akan terlihat seperti ini setelah Anda masuk mengikuti langkah-langkah yang diberikan di atas untuk Alat AWS untuk PowerShell atau AWS CLI.

```
[default]
login_session = arn:aws:iam::0123456789012:user/username
region = us-east-1

[my-dev-profile]
login_session = arn:aws:iam::0123456789012:user/username
region = us-east-1
```

Aktifkan dan konfigurasikan Pusat Identitas IAM

Untuk menggunakan IAM Identity Center, pertama-tama harus diaktifkan dan dikonfigurasi. Untuk melihat detail tentang cara melakukannya untuk SDK, lihat Langkah 1 dalam topik autentikasi Pusat Identitas IAM di Panduan Referensi Alat AWS SDKs dan Alat. Secara khusus, ikuti instruksi yang diperlukan di bawah Saya tidak memiliki akses yang ditetapkan melalui Pusat Identitas IAM.

Konfigurasikan SDK untuk menggunakan IAM Identity Center.

Informasi tentang cara mengonfigurasi SDK untuk menggunakan IAM Identity Center ada di Langkah 2 dalam topik autentikasi Pusat Identitas IAM di Panduan Referensi Alat AWS SDKs dan Alat. Setelah Anda menyelesaikan konfigurasi ini, sistem Anda harus berisi elemen-elemen berikut:

- Itu AWS CLI, yang Anda gunakan untuk memulai sesi portal AWS akses sebelum Anda menjalankan aplikasi Anda.
- AWS `config`File bersama yang berisi `[default]`profil dengan serangkaian nilai konfigurasi yang dapat direferensikan dari SDK. Untuk menemukan lokasi file ini, lihat Lokasi file bersama di Panduan Referensi Alat AWS SDKs dan. AWS SDK untuk .NET Menggunakan penyedia token SSO profil untuk memperoleh kredensil sebelum mengirim permintaan ke. AWS`sso_role_name`Nilai, yang merupakan peran IAM yang terhubung ke set izin Pusat Identitas IAM, harus memungkinkan akses ke yang Layanan AWS digunakan dalam aplikasi Anda.

`config`File contoh berikut menunjukkan profil default yang disiapkan dengan penyedia token SSO. `sso_session`Pengaturan profil mengacu pada `sso-session` bagian bernama. `sso-session`Bagian 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
```

Important

Jika Anda menggunakan AWS IAM Identity Center untuk otentikasi, aplikasi Anda harus mereferensikan NuGet paket-paket berikut agar resolusi SSO dapat berfungsi:

- `AWSSDK.SSO`
- `AWSSDK.SSOOIDC`

Kegagalan untuk mereferensikan paket-paket ini akan menghasilkan pengecualian runtime.

Mulai sesi portal AWS akses

Sebelum menjalankan aplikasi yang mengakses Layanan AWS, Anda memerlukan sesi portal AWS akses aktif agar SDK menggunakan autentikasi IAM Identity Center untuk menyelesaikan kredensialnya. 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
```

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

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

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

Respons terhadap perintah ini harus melaporkan akun IAM Identity Center 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 kredensil.
> 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 nama. `botocore`

Informasi tambahan

- Untuk informasi tambahan tentang penggunaan IAM Identity Center dan SSO di lingkungan pengembangan, lihat Sign-on tunggal di bagianAutentikasi tingkat lanjut. Informasi ini mencakup metode alternatif dan lebih canggih, serta tutorial yang menunjukkan kepada Anda cara menggunakan metode ini.
- Untuk opsi lainnya tentang otentikasi SDK, seperti penggunaan profil dan variabel lingkungan, lihat bagian konfigurasi di Panduan Referensi Alat AWS SDKs dan Alat.
- Untuk mempelajari lebih lanjut tentang praktik terbaik, lihat Praktik terbaik keamanan di IAM di Panduan Pengguna IAM.
- Untuk membuat AWS kredensil jangka pendek, lihat Kredenal Keamanan Sementara di Panduan Pengguna IAM.
- Untuk mempelajari tentang penyedia kredensi lainnya, lihat Penyedia kredensi terstandarisasi di Panduan Referensi Alat AWS SDKs dan Alat.

Membuat aplikasi sederhana menggunakan AWS SDK untuk .NET

Bagian ini menyediakan tutorial dasar untuk pengembang yang baru mengenal AWS SDK untuk .NET.

> **Note**
>
> Sebelum Anda menggunakan tutorial ini, Anda harus terlebih dahulu menginstal toolchain Anda dan mengkonfigurasi otentikasi SDK.

Untuk informasi tentang mengembangkan perangkat lunak untuk AWS layanan tertentu bersama dengan contoh kode, lihat AWS Layanan panggilan. Untuk contoh kode tambahan, lihatSDK untuk .NET (v4) contoh kode.

Topik

- Aplikasi lintas platform sederhana menggunakan AWS SDK untuk .NET
- Aplikasi berbasis Windows sederhana menggunakan AWS SDK untuk .NET
- Langkah selanjutnya

Aplikasi lintas platform sederhana menggunakan AWS SDK untuk .NET

Tutorial ini menggunakan AWS SDK untuk .NET dan .NET Core untuk pengembangan lintas platform. Tutorial ini menunjukkan cara menggunakan SDK untuk membuat daftar bucket Amazon S3 yang Anda miliki dan, secara opsional, membuat bucket.

Anda akan melakukan tutorial ini menggunakan alat lintas platform seperti antarmuka baris perintah .NET (CLI). Untuk cara lain untuk mengonfigurasi lingkungan pengembangan Anda, lihatMenginstal dan mengonfigurasi toolchain Anda untuk AWS SDK untuk .NET.

Diperlukan untuk pengembangan .NET lintas platform di Windows, Linux, atau macOS:

- Microsoft .NET Core SDK, versi 2.1, 3.1, atau yang lebih baru, yang mencakup antarmuka baris perintah .NET (CLI) **`dotnet`**() dan runtime .NET Core.

- Editor kode atau lingkungan pengembangan terintegrasi (IDE) yang sesuai untuk sistem operasi dan persyaratan Anda. Ini biasanya salah satu yang menyediakan beberapa dukungan untuk .NET Core.

  Contohnya termasuk Microsoft Visual Studio Code (VS Code), JetBrains Rider, dan Microsoft Visual Studio.

> **Note**
>
> Sebelum Anda menggunakan tutorial ini, Anda harus terlebih dahulu menginstal toolchain Anda dan mengkonfigurasi otentikasi SDK.

Langkah-langkah

- Buat proyek
- Buat kodenya
- Jalankan aplikasi
- Pembersihan

Buat proyek

1. Buka command prompt atau terminal. Temukan atau buat folder sistem operasi di mana Anda dapat membuat proyek.NET.
2. Dalam folder itu, jalankan perintah berikut untuk membuat proyek.NET.

   ```
   dotnet new console --name S3CreateAndList
   ```

3. Buka `S3CreateAndList` folder yang baru dibuat dan jalankan perintah berikut:

   ```
   dotnet add package AWSSDK.S3
   dotnet add package AWSSDK.SecurityToken
   dotnet add package AWSSDK.SSO
   dotnet add package AWSSDK.SSOOIDC
   ```

   Perintah sebelumnya menginstal NuGet paket dari manajer NuGet paket. Karena kita tahu persis NuGet paket apa yang kita butuhkan untuk tutorial ini, kita dapat melakukan langkah ini

sekarang. Ini juga umum bahwa paket yang diperlukan diketahui selama pengembangan. Ketika ini terjadi, perintah serupa dapat dijalankan pada saat itu.

Buat kodenya

1. Di `S3CreateAndList` folder, temukan dan buka `Program.cs` di editor kode Anda.
2. Ganti konten dengan kode berikut dan simpan file.

```
using System;
using System.Threading.Tasks;

// NuGet packages: AWSSDK.S3, AWSSDK.SecurityToken, AWSSDK.SSO, AWSSDK.SSOOIDC
using Amazon.Runtime;
using Amazon.Runtime.CredentialManagement;
using Amazon.S3;
using Amazon.S3.Model;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;

namespace S3CreateAndList
{
    class Program
    {
        // This code is part of the quick tour in the developer guide.
        // See https://docs.aws.amazon.com/sdk-for-net/v3/developer-guide/quick-
start.html
        // for complete steps.
        // Requirements:
        // - An SSO profile in the SSO user's shared config file with sufficient
 privileges for
  //   STS and S3 buckets.
        // - An active SSO Token.
        //    If an active SSO token isn't available, the SSO user should do the
 following:
        //    In a terminal, the SSO user must call "aws sso login".

        // Class members.
        static async Task Main(string[] args)
        {
            // Get SSO credentials from the information in the shared config file.
            // For this tutorial, the information is in the [default] profile.
            var ssoCreds = LoadSsoCredentials("default");
```

```
            // Display the caller's identity.
            var ssoProfileClient = new AmazonSecurityTokenServiceClient(ssoCreds);
            Console.WriteLine($"\nSSO Profile:\n {await
ssoProfileClient.GetCallerIdentityArn()}");

            // Create the S3 client is by using the SSO credentials obtained
earlier.
            var s3Client = new AmazonS3Client(ssoCreds);

            // Parse the command line arguments for the bucket name.
            if (GetBucketName(args, out String bucketName))
            {
                // If a bucket name was supplied, create the bucket.
                // Call the API method directly
                try
                {
                    Console.WriteLine($"\nCreating bucket {bucketName}...");
                    var createResponse = await s3Client.PutBucketAsync(bucketName);
                    Console.WriteLine($"Result:
{createResponse.HttpStatusCode.ToString()}");
                }
                catch (Exception e)
                {
                    Console.WriteLine("Caught exception when creating a bucket:");
                    Console.WriteLine(e.Message);
                }
            }

            // Display a list of the account's S3 buckets.
            Console.WriteLine("\nGetting a list of your buckets...");
            var listResponse = await s3Client.ListBucketsAsync();
            Console.WriteLine($"Number of buckets: {listResponse.Buckets.Count}");
            foreach (S3Bucket b in listResponse.Buckets)
            {
                Console.WriteLine(b.BucketName);
            }
            Console.WriteLine();
        }

        //
        // Method to parse the command line.
        private static Boolean GetBucketName(string[] args, out String bucketName)
        {
```

```
            Boolean retval = false;
            bucketName = String.Empty;
            if (args.Length == 0)
            {
                Console.WriteLine("\nNo arguments specified. Will simply list your
Amazon S3 buckets." +
                  "\nIf you wish to create a bucket, supply a valid, globally
unique bucket name.");
                bucketName = String.Empty;
                retval = false;
            }
            else if (args.Length == 1)
            {
                bucketName = args[0];
                retval = true;
            }
            else
            {
                Console.WriteLine("\nToo many arguments specified." +
                  "\n\ndotnet_tutorials - A utility to list your Amazon S3 buckets
and optionally create a new one." +
                  "\n\nUsage: S3CreateAndList [bucket_name]" +
                  "\n - bucket_name: A valid, globally unique bucket name." +
                  "\n - If bucket_name isn't supplied, this utility simply lists
your buckets.");
                Environment.Exit(1);
            }
            return retval;
        }

        //
        // Method to get SSO credentials from the information in the shared config
file.
        static AWSCredentials LoadSsoCredentials(string profile)
        {
            var chain = new CredentialProfileStoreChain();
            if (!chain.TryGetAWSCredentials(profile, out var credentials))
                throw new Exception($"Failed to find the {profile} profile");
            return credentials;
        }
    }

    // Class to read the caller's identity.
    public static class Extensions
```

```
    {
        public static async Task<string> GetCallerIdentityArn(this
 IAmazonSecurityTokenService stsClient)
        {
            var response = await stsClient.GetCallerIdentityAsync(new
 GetCallerIdentityRequest());
            return response.Arn;
        }
    }
}
```

Jalankan aplikasi

1. Jalankan perintah berikut.

```
dotnet run
```

2. Periksa output untuk melihat jumlah ember Amazon S3 yang Anda miliki, jika ada, dan namanya.
3. Pilih nama untuk ember Amazon S3 baru. Gunakan "dotnet-quicktour-s3-1-cross-” sebagai basis dan tambahkan sesuatu yang unik ke dalamnya, seperti GUID atau nama Anda. Pastikan untuk mengikuti aturan untuk nama bucket, seperti yang dijelaskan dalam Aturan untuk penamaan bucket di Panduan Pengguna Amazon S3.
4. Jalankan perintah berikut, ganti *amzn-s3-demo-bucket* dengan nama bucket yang Anda pilih.

```
dotnet run amzn-s3-demo-bucket
```

5. Periksa output untuk melihat bucket baru yang telah dibuat.

Pembersihan

Saat melakukan tutorial ini, Anda membuat beberapa sumber daya yang dapat Anda pilih untuk dibersihkan saat ini.

- Jika Anda tidak ingin menyimpan bucket yang dibuat aplikasi pada langkah sebelumnya, hapus dengan menggunakan konsol Amazon S3 di. https://console.aws.amazon.com/s3/
- Jika Anda tidak ingin menyimpan proyek .NET Anda, hapus `S3CreateAndList` folder dari lingkungan pengembangan Anda.

Ke mana harus pergi selanjutnya

Kembali ke menu tur cepat atau langsung ke akhir tur singkat ini.

Aplikasi berbasis Windows sederhana menggunakan AWS SDK untuk .NET

Tutorial ini menggunakan AWS SDK untuk .NET pada Windows dengan Visual Studio dan .NET Core. Tutorial ini menunjukkan cara menggunakan SDK untuk membuat daftar bucket Amazon S3 yang Anda miliki dan secara opsional membuat bucket.

Anda akan melakukan tutorial ini pada Windows menggunakan Visual Studio dan .NET Core. Untuk cara lain untuk mengonfigurasi lingkungan pengembangan Anda, lihatMenginstal dan mengonfigurasi toolchain Anda untuk AWS SDK untuk .NET.

Diperlukan untuk pengembangan pada Windows dengan Visual Studio dan .NET Core:

- Microsoft Visual Studio
- Microsoft .NET Core 2.1, 3.1 atau yang lebih baru

  Ini biasanya disertakan secara default saat menginstal versi terbaru Visual Studio.

> **Note**
>
> Sebelum Anda menggunakan tutorial ini, Anda harus terlebih dahulu menginstal toolchain Anda dan mengkonfigurasi otentikasi SDK.

Langkah-langkah

- Buat proyek
- Buat kodenya
- Jalankan aplikasi
- Pembersihan

Buat proyek

1. Buka Visual Studio dan buat proyek baru yang menggunakan versi C # dari template Aplikasi Konsol; yaitu, dengan deskripsi: "... untuk membuat aplikasi baris perintah yang dapat berjalan di .NET...". Beri nama proyek`S3CreateAndList`.

   > **Note**
   >
   > Jangan memilih versi .NET Framework dari template aplikasi konsol, atau, jika Anda melakukannya, pastikan untuk menggunakan .NET Framework 4.7.2 atau yang lebih baru.

2. Dengan proyek yang baru dibuat dimuat, pilih Tools, NuGetPackage Manager, Manage NuGet Packages for Solution.
3. Jelajahi NuGet paket-paket berikut dan instal ke dalam proyek:`AWSSDK.S3`,`AWSSDK.SecurityToken`,`AWSSDK.SSO`, dan `AWSSDK.SSOOIDC`

   Proses ini menginstal NuGet paket dari manajer NuGet paket. Karena kita tahu persis NuGet paket apa yang kita butuhkan untuk tutorial ini, kita dapat melakukan langkah ini sekarang. Ini juga umum bahwa paket yang diperlukan diketahui selama pengembangan. Ketika ini terjadi, ikuti proses serupa untuk menginstalnya pada saat itu.
4. Jika Anda bermaksud menjalankan aplikasi dari command prompt, buka command prompt sekarang dan navigasikan ke folder yang akan berisi output build. Ini biasanya sesuatu seperti`S3CreateAndList\S3CreateAndList\bin\Debug\net6.0`, tetapi akan tergantung pada lingkungan Anda.

Buat kodenya

1. Dalam `S3CreateAndList` proyek, temukan dan buka `Program.cs` di IDE.
2. Ganti konten dengan kode berikut dan simpan file.

```
using System;
using System.Threading.Tasks;

// NuGet packages: AWSSDK.S3, AWSSDK.SecurityToken, AWSSDK.SSO, AWSSDK.SSOOIDC
using Amazon.Runtime;
using Amazon.Runtime.CredentialManagement;
using Amazon.S3;
```

```
using Amazon.S3.Model;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;

namespace S3CreateAndList
{
    class Program
    {
        // This code is part of the quick tour in the developer guide.
        // See https://docs.aws.amazon.com/sdk-for-net/v3/developer-guide/quick-
start.html
        // for complete steps.
        // Requirements:
        // - An SSO profile in the SSO user's shared config file with sufficient
 privileges for
  //   STS and S3 buckets.
        // - An active SSO Token.
        //    If an active SSO token isn't available, the SSO user should do the
 following:
        //    In a terminal, the SSO user must call "aws sso login".

        // Class members.
        static async Task Main(string[] args)
        {
            // Get SSO credentials from the information in the shared config file.
            // For this tutorial, the information is in the [default] profile.
            var ssoCreds = LoadSsoCredentials("default");

            // Display the caller's identity.
            var ssoProfileClient = new AmazonSecurityTokenServiceClient(ssoCreds);
            Console.WriteLine($"\nSSO Profile:\n {await
 ssoProfileClient.GetCallerIdentityArn()}");

            // Create the S3 client is by using the SSO credentials obtained
 earlier.
            var s3Client = new AmazonS3Client(ssoCreds);

            // Parse the command line arguments for the bucket name.
            if (GetBucketName(args, out String bucketName))
            {
                // If a bucket name was supplied, create the bucket.
                // Call the API method directly
                try
                {
```

```
                Console.WriteLine($"\nCreating bucket {bucketName}...");
                var createResponse = await s3Client.PutBucketAsync(bucketName);
                Console.WriteLine($"Result:
{createResponse.HttpStatusCode.ToString()}");
            }
            catch (Exception e)
            {
                Console.WriteLine("Caught exception when creating a bucket:");
                Console.WriteLine(e.Message);
            }
        }

        // Display a list of the account's S3 buckets.
        Console.WriteLine("\nGetting a list of your buckets...");
        var listResponse = await s3Client.ListBucketsAsync();
        Console.WriteLine($"Number of buckets: {listResponse.Buckets.Count}");
        foreach (S3Bucket b in listResponse.Buckets)
        {
            Console.WriteLine(b.BucketName);
        }
        Console.WriteLine();
    }

    //
    // Method to parse the command line.
    private static Boolean GetBucketName(string[] args, out String bucketName)
    {
        Boolean retval = false;
        bucketName = String.Empty;
        if (args.Length == 0)
        {
            Console.WriteLine("\nNo arguments specified. Will simply list your
Amazon S3 buckets." +
              "\nIf you wish to create a bucket, supply a valid, globally
unique bucket name.");
            bucketName = String.Empty;
            retval = false;
        }
        else if (args.Length == 1)
        {
            bucketName = args[0];
            retval = true;
        }
        else
```

```
            {
                Console.WriteLine("\nToo many arguments specified." +
                  "\n\ndotnet_tutorials - A utility to list your Amazon S3 buckets
 and optionally create a new one." +
                  "\n\nUsage: S3CreateAndList [bucket_name]" +
                  "\n - bucket_name: A valid, globally unique bucket name." +
                  "\n - If bucket_name isn't supplied, this utility simply lists
 your buckets.");
                Environment.Exit(1);
            }
            return retval;
        }

        //
        // Method to get SSO credentials from the information in the shared config
 file.
        static AWSCredentials LoadSsoCredentials(string profile)
        {
            var chain = new CredentialProfileStoreChain();
            if (!chain.TryGetAWSCredentials(profile, out var credentials))
                throw new Exception($"Failed to find the {profile} profile");
            return credentials;
        }
    }

    // Class to read the caller's identity.
    public static class Extensions
    {
        public static async Task<string> GetCallerIdentityArn(this
 IAmazonSecurityTokenService stsClient)
        {
            var response = await stsClient.GetCallerIdentityAsync(new
 GetCallerIdentityRequest());
            return response.Arn;
        }
    }
}
```

3. Bangun aplikasi.

> **Note**
>
> Jika Anda menggunakan versi Visual Studio yang lebih lama, Anda mungkin mendapatkan error build yang mirip dengan berikut ini:
> “Fitur 'async main' tidak tersedia di C # 7.0. Silakan gunakan bahasa versi 7.1 atau lebih tinggi.”
> Jika Anda mendapatkan kesalahan ini, siapkan proyek Anda untuk menggunakan versi bahasa yang lebih baru. Ini biasanya dilakukan di properti proyek, Build, Advanced.

Jalankan aplikasi

1. Jalankan aplikasi tanpa argumen baris perintah. Lakukan ini baik di command prompt (jika Anda membukanya sebelumnya) atau dari IDE.
2. Periksa output untuk melihat jumlah ember Amazon S3 yang Anda miliki, jika ada, dan namanya.
3. Pilih nama untuk ember Amazon S3 baru. Gunakan "dotnet-quicktour-s3-1-winvs-” sebagai basis dan tambahkan sesuatu yang unik ke dalamnya, seperti GUID atau nama Anda. Pastikan untuk mengikuti aturan untuk nama bucket, seperti yang dijelaskan dalam Aturan untuk Penamaan Bucket di Panduan Pengguna Amazon S3.
4. Jalankan aplikasi lagi, kali ini memasok nama bucket.

   Di baris perintah, ganti *amzn-s3-demo-bucket* perintah berikut dengan nama bucket yang Anda pilih.

   ```
   S3CreateAndList amzn-s3-demo-bucket
   ```

   Atau, jika Anda menjalankan aplikasi di IDE, pilih Project, S3 CreateAndList Properties, Debug dan masukkan nama bucket di sana.
5. Periksa output untuk melihat bucket baru yang telah dibuat.

Pembersihan

Saat melakukan tutorial ini, Anda membuat beberapa sumber daya yang dapat Anda pilih untuk dibersihkan saat ini.

- Jika Anda tidak ingin menyimpan bucket yang dibuat aplikasi pada langkah sebelumnya, hapus dengan menggunakan konsol Amazon S3 di. https://console.aws.amazon.com/s3/
- Jika Anda tidak ingin menyimpan proyek .NET Anda, hapus `S3CreateAndList` folder dari lingkungan pengembangan Anda.

Ke mana harus pergi selanjutnya

Kembali ke menu tur cepat atau langsung ke akhir tur singkat ini.

Langkah selanjutnya

Pastikan untuk membersihkan sumber daya sisa yang Anda buat saat melakukan tutorial ini. Ini mungkin AWS sumber daya atau sumber daya di lingkungan pengembangan Anda seperti file dan folder.

Sekarang setelah Anda melakukan tur AWS SDK untuk .NET, Anda mungkin ingin memulai proyek Anda.

Mengkonfigurasi AWS SDK untuk .NET

Untuk menggunakannya AWS SDK untuk .NET, Anda perlu mengatur lingkungan Anda dan juga mengonfigurasi sejumlah hal penting yang dibutuhkan aplikasi Anda untuk mengakses AWS layanan. Ini termasuk:

- Spesifikasi AWS Daerah
- AWSSDK paket atau rakitan

Beberapa topik di bagian ini memberikan informasi tentang cara mengonfigurasi hal-hal penting ini.

Topik lain di bagian ini dan bagian lain memberikan informasi tentang cara yang lebih maju yang dapat Anda konfigurasi proyek Anda.

Topik

- Memulai proyek baru
- Instal AWSSDK paket dengan NuGet
- Instal AWSSDK rakitan tanpa NuGet
- Mengatur AWS Wilayah untuk AWS SDK untuk .NET
- Resolusi kredensi dan profil
- Mencoba lagi dan batas waktu
- Observabilitas
- Informasi tambahan tentang pengguna dan peran
- Konfigurasi lanjutan untuk AWS SDK untuk .NET proyek Anda
- Menggunakan kredensial warisan

Memulai proyek baru

Ada beberapa teknik yang dapat Anda gunakan untuk memulai proyek baru untuk mengakses AWS layanan. Berikut ini adalah beberapa teknik tersebut:

- Jika Anda baru mengenal pengembangan.NET AWS atau setidaknya baru mengenal AWS SDK untuk .NET, Anda dapat melihat contoh lengkapnya diMembuat aplikasi sederhana. Ini memberi Anda pengantar SDK.

- Anda dapat memulai proyek dasar dengan menggunakan .NET CLI. Untuk melihat contoh ini, buka prompt perintah atau terminal, buat folder atau direktori dan arahkan ke sana, lalu masukkan yang berikut ini.

  ```
  dotnet new console --name [SOME-NAME]
  ```

  Proyek kosong dibuat di mana Anda dapat menambahkan kode dan NuGet paket. Untuk informasi selengkapnya, lihat panduan .NET Core.

  Untuk melihat daftar templat proyek, gunakan yang berikut ini: `dotnet new --list`

- AWS Toolkit for Visual Studio Termasuk template proyek C # untuk berbagai AWS layanan. Setelah Anda menginstal toolkit di Visual Studio, Anda dapat mengakses template saat membuat proyek baru.

  Untuk melihat ini, buka Bekerja dengan AWS layanan di Panduan AWS Toolkit for Visual Studio Pengguna. Beberapa contoh di bagian itu membuat proyek baru.

- Jika Anda mengembangkan dengan Visual Studio di Windows tetapi tanpa AWS Toolkit for Visual Studio, gunakan teknik khas Anda untuk membuat proyek baru.

  Untuk melihat contoh, buka Visual Studio dan pilih File, New, Project. Cari ".net core" dan pilih versi C # dari template Console App (.NET Core) atau WPF App (.NET Core). Proyek kosong dibuat di mana Anda dapat menambahkan kode dan NuGet paket.

Anda dapat menemukan beberapa contoh cara bekerja dengan AWS layanan diContoh kode terpandu.

> **Important**
>
> Jika Anda menggunakan AWS IAM Identity Center untuk otentikasi, aplikasi Anda harus mereferensikan NuGet paket-paket berikut agar resolusi SSO dapat berfungsi:
>
> - `AWSSDK.SSO`
> - `AWSSDK.SSOOIDC`

Kegagalan untuk mereferensikan paket-paket ini akan menghasilkan pengecualian runtime.

Instal AWSSDK paket dengan NuGet

NuGetadalah sistem manajemen paket untuk platform.NET. Dengan NuGet, Anda dapat menginstal AWSSDKpaket, serta beberapa ekstensi lainnya, ke proyek Anda. Untuk informasi tambahan, lihat repositori aws/dotnet di situs web. GitHub

NuGet selalu memiliki versi terbaru dari AWSSDK paket, serta versi sebelumnya. NuGetmengetahui ketergantungan antara paket dan menginstal semua paket yang diperlukan secara otomatis.

Warning

Daftar NuGet paket mungkin menyertakan satu bernama hanya "AWSSDK" (tanpa pengenal yang ditambahkan). JANGAN menginstal NuGet paket ini; itu warisan dan tidak boleh digunakan untuk proyek baru.

Paket yang diinstal dengan NuGet disimpan dengan proyek Anda, bukan di lokasi pusat. Ini memungkinkan Anda untuk menginstal versi perakitan khusus untuk aplikasi tertentu tanpa membuat masalah kompatibilitas untuk aplikasi lain. Untuk informasi selengkapnya NuGet, lihat NuGet dokumentasi.

Note

Jika Anda tidak dapat atau tidak diizinkan untuk mengunduh dan menginstal NuGet paket per proyek, Anda dapat memperoleh AWSSDK rakitan dan menyimpannya secara lokal (atau di tempat).
Jika ini berlaku untuk Anda dan Anda belum mendapatkan AWSSDK majelis, lihat. Memperoleh AWSSDK majelis Untuk mempelajari cara menggunakan rakitan yang disimpan secara lokal, lihat. Instal AWSSDK rakitan tanpa NuGet

Menggunakan NuGet dari Command prompt atau terminal

1. Buka paket NuGet dan tentukan AWSSDK paket mana yang Anda butuhkan dalam proyek Anda; misalnya, AWSSDK.S3.
2. Salin perintah.NET CLI dari halaman web paket itu, seperti yang ditunjukkan pada contoh berikut.

   ```
   dotnet add package AWSSDK.S3 --version 3.3.110.19
   ```

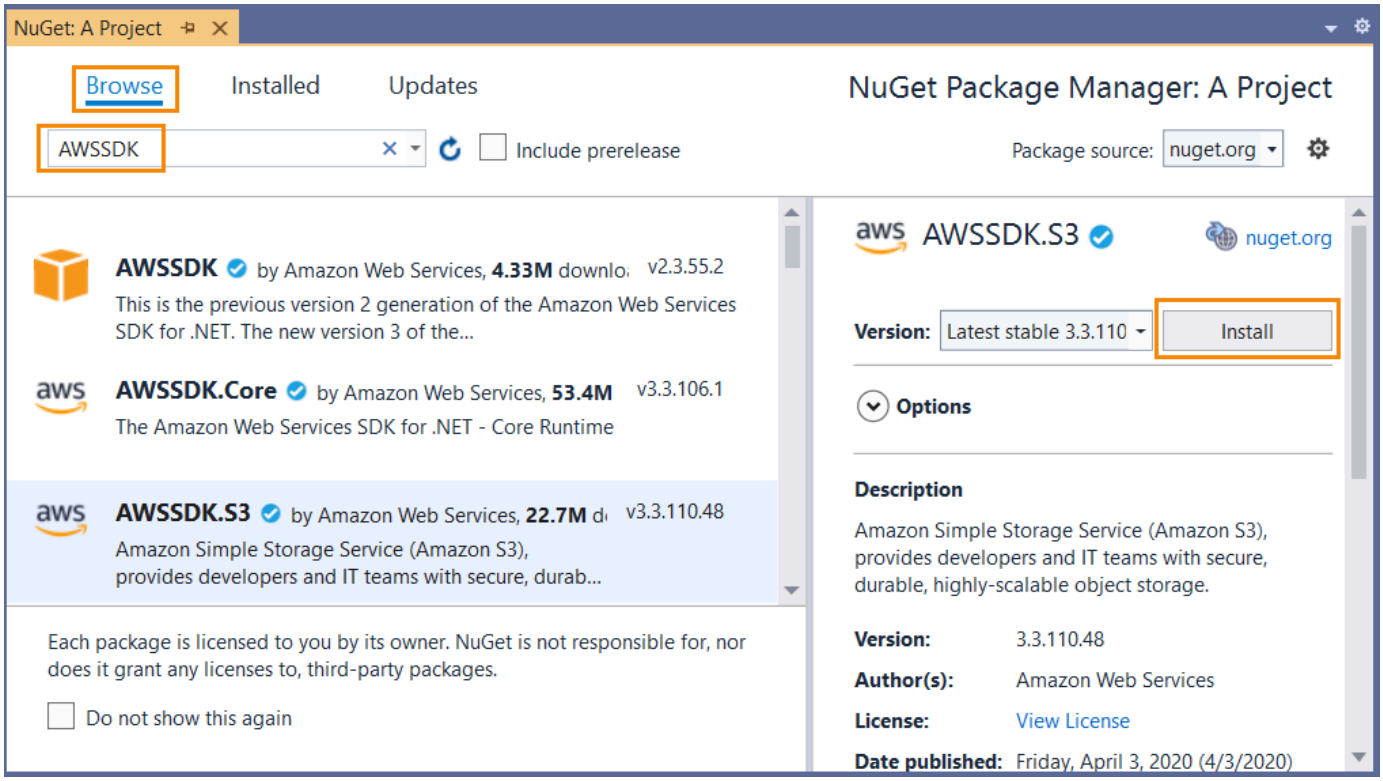
3. Di direktori proyek Anda, jalankan perintah.NET CLI itu. NuGet juga menginstal dependensi apa pun, seperti .Core. AWSSDK

> **Note**
>
> Jika Anda hanya menginginkan versi terbaru dari sebuah NuGet paket, Anda dapat mengecualikan informasi versi dari perintah, seperti yang ditunjukkan pada contoh berikut.
> `dotnet add package AWSSDK.S3`

Menggunakan NuGet dari Visual Studio Solution Explorer

1. Di Solution Explorer, klik kanan proyek Anda, lalu pilih Kelola NuGet Paket dari menu konteks.
2. Di panel kiri NuGet Package Manager, pilih Browse. Anda kemudian dapat menggunakan kotak pencarian untuk mencari paket yang ingin Anda instal. NuGet juga menginstal dependensi apa pun, seperti .Core. AWSSDK

   Gambar berikut menunjukkan instalasi AWSSDKpaket.S3.



Menggunakan NuGet dari Package Manager Console

Di Visual Studio, pilih Tools, NuGet Package Manager, Package Manager Console.

Anda dapat menginstal AWSSDK paket yang Anda inginkan dari Package Manager Console dengan menggunakan **`Install-Package`**perintah. Misalnya, untuk AWSSDKmenginstal.S3, gunakan perintah berikut.

```
PM> Install-Package AWSSDK.S3
```

NuGet juga menginstal dependensi apa pun, seperti .Core. AWSSDK

Jika Anda perlu menginstal versi paket yang lebih lama, gunakan `-Version` opsi dan tentukan versi paket yang Anda inginkan, seperti yang ditunjukkan pada contoh berikut.

```
PM> Install-Package AWSSDK.S3 -Version 3.3.106.6
```

Untuk informasi selengkapnya tentang perintah Package Manager Console, lihat PowerShellreferensi di NuGetdokumentasi Microsoft.

Instal AWSSDK rakitan tanpa NuGet

Topik ini menjelaskan bagaimana Anda dapat menggunakan AWSSDK rakitan yang Anda peroleh dan simpan secara lokal (atau di tempat) seperti yang dijelaskan dalam. Memperoleh AWSSDK majelis Ini bukan metode yang direkomendasikan untuk menangani referensi SDK, tetapi diperlukan di beberapa lingkungan.

> **Note**
>
> Metode yang disarankan untuk menangani referensi SDK adalah mengunduh dan menginstal hanya NuGet paket yang dibutuhkan setiap proyek. Metode itu dijelaskan dalamInstal AWSSDK paket dengan NuGet.

Untuk menginstal AWSSDK rakitan

1. Buat folder di area proyek Anda untuk AWSSDK rakitan yang diperlukan. Sebagai contoh, Anda dapat memanggil folder ini`AwsAssemblies`.
2. Jika Anda belum melakukannya, dapatkan rakitan, yang menempatkan AWSSDK rakitan di beberapa folder unduhan atau instalasi lokal. Salin file DLL untuk rakitan yang diperlukan dari folder unduhan itu ke proyek Anda (ke dalam `AwsAssemblies` folder, dalam contoh kami).

   Pastikan juga untuk menyalin dependensi apa pun. Anda dapat menemukan informasi tentang dependensi di situs web. GitHub
3. Buat referensi ke rakitan yang diperlukan sebagai berikut.

   Cross-platform development

   1. Buka `.csproj` file proyek Anda dan tambahkan `<ItemGroup>` elemen.
   2. Dalam `<ItemGroup>` elemen, tambahkan `<Reference>` elemen dengan `Include` atribut untuk setiap rakitan yang diperlukan.

      Untuk Amazon S3, misalnya, Anda akan menambahkan baris berikut ke file proyek Anda. `.csproj`

      Di Linux dan macOS:

      ```
      <ItemGroup>
      ```

```
  <Reference Include="./AwsAssemblies/AWSSDK.Core.dll" />
  <Reference Include="./AwsAssemblies/AWSSDK.S3.dll" />
</ItemGroup>
```

Di Windows:

```
<ItemGroup>
  <Reference Include="AwsAssemblies\AWSSDK.Core.dll" />
  <Reference Include="AwsAssemblies\AWSSDK.S3.dll" />
</ItemGroup>
```

3. Simpan `.csproj` file proyek Anda.

Windows with Visual Studio and .NET Core

1. Di Visual Studio, muat proyek Anda dan buka Proyek, Tambahkan Referensi.
2. Pilih tombol Browse di bagian bawah kotak dialog. Arahkan ke folder proyek Anda dan subfolder tempat Anda menyalin file DLL yang diperlukan (`AwsAssemblies`, misalnya).
3. Pilih semua file DLL, pilih Tambah, dan pilih OK.
4. Simpan proyek Anda.

Mengatur AWS Wilayah untuk AWS SDK untuk .NET

AWS Wilayah memungkinkan Anda mengakses AWS layanan yang secara fisik berada di wilayah geografis tertentu. Ini dapat berguna untuk redundansi dan untuk menjaga data dan aplikasi Anda berjalan dekat dengan tempat Anda dan pengguna Anda akan mengaksesnya.

Untuk melihat daftar saat ini dari semua Wilayah dan titik akhir yang didukung untuk setiap AWS layanan, lihat Titik akhir dan kuota layanan di. Referensi Umum AWS Untuk melihat daftar titik akhir Regional yang ada, lihat titik akhir AWS layanan. Untuk melihat informasi terperinci tentang Wilayah, lihat Menentukan AWS Wilayah mana yang dapat digunakan akun Anda.

Anda dapat membuat klien AWS layanan yang masuk ke Wilayah tertentu. Anda juga dapat mengonfigurasi aplikasi Anda dengan Wilayah yang akan digunakan untuk semua klien AWS layanan. Kedua kasus ini dijelaskan selanjutnya.

Buat klien layanan dengan Wilayah tertentu

Anda dapat menentukan Wilayah untuk salah satu klien AWS layanan dalam aplikasi Anda. Menyetel Wilayah dengan cara ini lebih diutamakan daripada pengaturan global apa pun untuk klien layanan tertentu.

Wilayah yang Ada

Contoh ini menunjukkan kepada Anda cara membuat instance EC2 klien Amazon di Wilayah yang ada. Ini menggunakan RegionEndpointbidang yang ditentukan.

```
using (AmazonEC2Client ec2Client = new AmazonEC2Client(RegionEndpoint.USWest2))
{
  // Make a request to EC2 in the us-west-2 Region using ec2Client
}
```

Wilayah Baru menggunakan RegionEndpoint kelas

Contoh ini menunjukkan kepada Anda bagaimana membangun titik akhir Region baru dengan menggunakan. RegionEndpoint GetBySystemName.

```
var newRegion = RegionEndpoint.GetBySystemName("us-west-new");
using (var ec2Client = new AmazonEC2Client(newRegion))
{
  // Make a request to EC2 in the new Region using ec2Client
}
```

Wilayah Baru menggunakan kelas konfigurasi klien layanan

Contoh ini menunjukkan cara menggunakan `ServiceURL` properti kelas konfigurasi klien layanan untuk menentukan Wilayah; dalam hal ini, menggunakan kelas Amazon EC2 Config.

Teknik ini berfungsi bahkan jika titik akhir Wilayah tidak mengikuti pola titik akhir Wilayah biasa.

```
var ec2ClientConfig = new AmazonEC2Config
{
    // Specify the endpoint explicitly
    ServiceURL = "https://ec2.us-west-new.amazonaws.com"
};
```

```
using (var ec2Client = new AmazonEC2Client(ec2ClientConfig))
{
  // Make a request to EC2 in the new Region using ec2Client
}
```

Tentukan Wilayah untuk semua klien layanan

Ada beberapa cara Anda dapat menentukan Wilayah untuk semua klien AWS layanan yang dibuat aplikasi Anda. Wilayah ini digunakan untuk klien layanan yang tidak dibuat dengan Wilayah tertentu.

Mencari AWS SDK untuk .NET nilai Region dalam urutan berikut.

Profil

Setel di profil yang telah dimuat aplikasi atau SDK Anda. Untuk informasi selengkapnya, lihat Resolusi kredensi dan profil.

Variabel-variabel lingkungan

Ditetapkan dalam variabel `AWS_REGION` lingkungan.

Di Linux atau macOS:

```
export AWS_REGION='us-west-2'
```

Di Windows:

```
set AWS_REGION=us-west-2
```

> **Note**
>
> Jika Anda menyetel variabel lingkungan ini untuk seluruh sistem (menggunakan `export` atau`setx`), itu memengaruhi semua SDKs dan toolkit, bukan hanya. AWS SDK untuk .NET

AWSConfigs kelas

Tetapkan sebagai AWSConfigs. AWSRegionproperti.

```
AWSConfigs.AWSRegion = "us-west-2";
using (var ec2Client = new AmazonEC2Client())
{
  // Make request to Amazon EC2 in us-west-2 Region using ec2Client
}
```

Resolusi wilayah

Jika tidak ada metode yang dijelaskan di atas yang digunakan untuk menentukan Wilayah AWS, AWS SDK untuk .NET upaya untuk menemukan Wilayah untuk klien AWS layanan untuk beroperasi.

Urutan resolusi wilayah

1. File konfigurasi aplikasi seperti `app.config` dan`web.config`.
2. Variabel lingkungan (`AWS_REGION`dan`AWS_DEFAULT_REGION`).
3. Profil dengan nama yang ditentukan oleh nilai di`AWSConfigs.AWSProfileName`.
4. Profil dengan nama yang ditentukan oleh variabel `AWS_PROFILE` lingkungan.
5. `[default]`Profil.
6. Metadata EC2 instans Amazon (jika berjalan pada sebuah EC2 instance).

Jika tidak ada Region yang ditemukan, SDK akan menampilkan pengecualian yang menyatakan bahwa klien AWS layanan tidak memiliki Region yang dikonfigurasi.

Informasi khusus tentang Wilayah Tiongkok (Beijing)

Untuk menggunakan layanan di Wilayah Tiongkok (Beijing), Anda harus memiliki akun dan kredensil yang khusus untuk Wilayah Tiongkok (Beijing). Akun dan kredensil untuk AWS Wilayah lain tidak akan berfungsi untuk Wilayah Tiongkok (Beijing). Demikian juga, akun dan kredensil untuk Wilayah Tiongkok (Beijing) tidak akan berfungsi untuk Wilayah lain. AWS Untuk informasi tentang titik akhir dan protokol yang tersedia di Wilayah Tiongkok (Beijing), lihat Titik Akhir Wilayah Beijing.

Informasi khusus tentang AWS layanan baru

AWS Layanan baru dapat diluncurkan pada awalnya di beberapa Wilayah dan kemudian didukung di Wilayah lain. Dalam kasus ini, Anda tidak perlu menginstal SDK terbaru untuk mengakses Wilayah baru untuk layanan tersebut. Anda dapat menentukan Wilayah yang baru ditambahkan berdasarkan per klien atau secara global, seperti yang ditunjukkan sebelumnya.

Resolusi kredensi dan profil

AWS SDK untuk .NET Pencarian kredensil dalam urutan tertentu dan menggunakan set pertama yang tersedia untuk aplikasi saat ini.

Urutan pencarian kredensi

1. Kredensil yang ditetapkan secara eksplisit pada klien AWS layanan, seperti yang dijelaskan dalam. Mengakses kredensi dan profil dalam aplikasi

   > **Note**
   >
   > Topik itu ada di Pertimbangan khusus bagian karena itu bukan metode yang disukai untuk menentukan kredensil.

2. Sesi AWSCredentials yang dibuat dari`AWS_ACCESS_KEY_ID`,`AWS_SECRET_ACCESS_KEY`, dan variabel `AWS_SESSION_TOKEN` lingkungan, jika ketiga variabel memiliki nilai.
3. Dasar AWSCredentials yang dibuat dari variabel `AWS_ACCESS_KEY_ID` dan `AWS_SECRET_ACCESS_KEY` lingkungan, jika kedua variabel memiliki nilai.
4. AssumeRoleWithWebIdentityCredentialsyang dibuat dari variabel `AWS_WEB_IDENTITY_TOKEN_FILE` dan `AWS_ROLE_ARN` lingkungan, jika kedua variabel memiliki nilai.
5. Profil kredensil dengan nama yang ditentukan oleh nilai di. AWSConfigs AWSProfileNama.
6. Profil kredensil dengan nama yang ditentukan oleh variabel `AWS_PROFILE` lingkungan.
7. Profil `[default]` kredensialnya.
8. Penyedia kredensi kontainer.
9. EC2 Metadata instans Amazon.

Jika aplikasi Anda berjalan di EC2 instans Amazon, seperti di lingkungan produksi, gunakan peran IAM seperti yang dijelaskan dalamMemberikan akses dengan menggunakan peran IAM. Jika tidak, seperti dalam pengujian prarilis, simpan kredensil Anda dalam file yang menggunakan format file AWS kredensial yang dapat diakses aplikasi web Anda di server.

Untuk informasi tambahan tentang variabel lingkungan dan profil kredensial, lihat topik berikut di Panduan Referensi Alat AWS SDKs dan Alat: Variabel lingkungan, daftar variabel Lingkungan, serta file konfigurasi dan kredensial bersama.

Resolusi profil

Dengan dua mekanisme penyimpanan yang berbeda untuk kredensil, penting untuk memahami cara mengonfigurasi AWS SDK untuk .NET untuk menggunakannya. The AWSConfigs. AWSProfilesProperti lokasi mengontrol cara AWS SDK untuk .NET menemukan profil kredensi.

AWSProfilesLokasi	Perilaku resolusi profil
nihil (tidak diatur) atau kosong	Cari SDK Store jika platform mendukungnya, lalu cari file AWS kredensi bersama di lokasi default. Jika profil tidak berada di salah satu lokasi tersebut, cari `~/.aws/config` (Linux atau macOS) atau `%USERPROFILE%\.aws\config` (Windows).
Path ke file dalam format file AWS kredensial	Cari hanya file yang ditentukan untuk profil dengan nama yang ditentukan.

Menggunakan kredensil akun pengguna federasi

Aplikasi yang menggunakan AWS SDK untuk .NET (AWSSDK.Core versi 3.1.6.0 dan yang lebih baru) dapat menggunakan akun pengguna federasi melalui Active Directory Federation Services (AD FS) untuk mengakses AWS layanan dengan menggunakan Security Assertion Markup Language (SAFL).

Dukungan akses federasi berarti pengguna dapat mengautentikasi menggunakan Active Directory Anda. Kredensi sementara diberikan kepada pengguna secara otomatis. Kredensi sementara ini, yang berlaku selama satu jam, digunakan saat aplikasi Anda memanggil AWS layanan. SDK menangani pengelolaan kredensi sementara. Untuk akun pengguna yang bergabung dengan domain, jika aplikasi Anda melakukan panggilan tetapi kredensialnya telah kedaluwarsa, pengguna akan diautentikasi ulang secara otomatis dan kredensi baru diberikan. (Untuk akun yang tidak bergabung dengan domain, pengguna diminta untuk memasukkan kredensil sebelum autentikasi ulang.)

Untuk menggunakan dukungan ini di aplikasi.NET Anda, Anda harus terlebih dahulu mengatur profil peran dengan menggunakan PowerShell cmdlet. Untuk mempelajari caranya, lihat AWS Tools for Windows PowerShell dokumentasi.

Setelah Anda mengatur profil peran, rujuk profil di aplikasi Anda. Ada beberapa cara untuk melakukan ini, salah satunya adalah dengan menggunakan AWSConfigs. AWSProfileBeri nama properti dengan cara yang sama seperti yang Anda lakukan dengan profil kredensi lainnya.

AWS Security Token ServiceMajelis (AWSSDK. SecurityToken) menyediakan dukungan SAFL untuk mendapatkan AWS kredensil. Untuk menggunakan kredensil akun pengguna federasi, pastikan perakitan ini tersedia untuk aplikasi Anda.

Menentukan peran atau kredensi sementara

Untuk aplikasi yang berjalan di EC2 instans Amazon, cara paling aman untuk mengelola kredensil adalah dengan menggunakan peran IAM, seperti yang dijelaskan dalam. Memberikan akses dengan menggunakan peran IAM

Untuk skenario aplikasi di mana perangkat lunak yang dapat dieksekusi tersedia untuk pengguna di luar organisasi Anda, kami sarankan Anda merancang perangkat lunak untuk menggunakan kredensil keamanan sementara. Selain menyediakan akses terbatas ke AWS sumber daya, kredensil ini memiliki manfaat kedaluwarsa setelah jangka waktu tertentu. Untuk informasi selengkapnya tentang kredensil keamanan sementara, lihat berikut ini:

- Kredensil keamanan sementara
- Kumpulan identitas Amazon Cognito

Menggunakan kredensil proxy

Jika perangkat lunak Anda berkomunikasi dengan AWS melalui proxy, Anda dapat menentukan kredensional untuk proxy dengan menggunakan `ProxyCredentials` properti `Config` kelas layanan. `Config`Kelas layanan biasanya merupakan bagian dari namespace utama untuk layanan. Contohnya termasuk yang berikut: AmazonCloudDirectoryConfigdi Amazon. CloudDirectorynamespace dan AmazonGameLiftConfigdi Amazon. GameLiftnamespace.

Untuk Amazon S3, misalnya, Anda dapat menggunakan kode yang mirip dengan berikut ini, di mana `SecurelyStoredUserName` dan `SecurelyStoredPassword` merupakan nama pengguna proxy dan kata sandi yang NetworkCredentialditentukan dalam objek.

```
AmazonS3Config config = new AmazonS3Config();
config.ProxyCredentials = new NetworkCredential(SecurelyStoredUserName,
 SecurelyStoredPassword);
```

> **Note**
>
> Versi SDK sebelumnya digunakan `ProxyUsername` dan`ProxyPassword`, tetapi properti ini tidak digunakan lagi.

Mencoba lagi dan batas waktu

AWS SDK untuk .NET Ini memungkinkan Anda untuk mengonfigurasi jumlah percobaan ulang dan nilai batas waktu untuk permintaan HTTP ke AWS layanan. Jika nilai default untuk percobaan ulang dan batas waktu tidak sesuai untuk aplikasi Anda, Anda dapat menyesuaikannya dengan persyaratan spesifik Anda, tetapi penting untuk memahami bagaimana hal itu akan memengaruhi perilaku aplikasi Anda.

Untuk menentukan nilai mana yang akan digunakan untuk percobaan ulang dan batas waktu, pertimbangkan hal berikut:

- Bagaimana seharusnya AWS SDK untuk .NET dan aplikasi Anda merespons ketika konektivitas jaringan menurun atau AWS layanan tidak dapat dijangkau? Apakah Anda ingin panggilan gagal dengan cepat, atau apakah pantas untuk panggilan untuk terus mencoba lagi atas nama Anda?
- Apakah aplikasi Anda merupakan aplikasi atau situs web yang menghadap pengguna yang harus responsif, atau apakah itu pekerjaan pemrosesan latar belakang yang memiliki toleransi lebih untuk peningkatan latensi?
- Apakah aplikasi digunakan pada jaringan yang andal dengan latensi rendah, atau apakah itu digunakan di lokasi terpencil dengan konektivitas yang tidak dapat diandalkan?

Percobaan ulang

Ikhtisar

Permintaan AWS SDK untuk .NET dapat mencoba ulang yang gagal karena pelambatan sisi server atau koneksi terputus. Ada dua properti kelas konfigurasi layanan yang dapat Anda gunakan untuk menentukan perilaku coba lagi dari klien layanan. Kelas konfigurasi layanan mewarisi properti ini dari Amazon.Runtime abstrak. ClientConfigkelas Referensi AWS SDK untuk .NET API:

- `RetryMode`menentukan salah satu dari tiga mode coba lagi, yang didefinisikan dalam Amazon.Runtime. RequestRetryModepencacahan.

Nilai default untuk aplikasi Anda dapat dikontrol dengan menggunakan variabel `AWS_RETRY_MODE` lingkungan atau pengaturan retry_mode dalam file konfigurasi bersama AWS .

- `MaxErrorRetry`menentukan jumlah percobaan ulang yang diizinkan di tingkat klien layanan; SDK mencoba ulang operasi beberapa kali yang ditentukan sebelum gagal dan melempar pengecualian.

  Nilai default untuk aplikasi Anda dapat dikontrol dengan menggunakan variabel `AWS_MAX_ATTEMPTS` lingkungan atau pengaturan max_attempts dalam file AWS konfigurasi bersama.

Deskripsi terperinci untuk properti ini dapat ditemukan di Amazon.Runtime abstrak. ClientConfigkelas Referensi AWS SDK untuk .NET API. Setiap nilai `RetryMode` sesuai secara default dengan nilai tertentu`MaxErrorRetry`, seperti yang ditunjukkan pada tabel berikut.

RetryMode	Sesuai MaxErrorRetry (Amazon DynamoDB)	Sesuai MaxErrorRetry (semua yang lain)
Warisan	10	4
Standar	10	2
Adaptif (eksperimental)	10	2

Perilaku

Saat aplikasi Anda dimulai

Ketika aplikasi Anda dimulai, nilai default untuk `RetryMode` dan `MaxErrorRetry` dikonfigurasi oleh SDK. Nilai default ini digunakan saat Anda membuat klien layanan kecuali Anda menentukan nilai lainnya.

- Jika properti tidak disetel di lingkungan Anda, default untuk `RetryMode` dikonfigurasi sebagai Legacy dan default untuk `MaxErrorRetry` dikonfigurasi dengan nilai yang sesuai dari tabel sebelumnya.
- Jika mode coba lagi telah disetel di lingkungan Anda, nilai tersebut digunakan sebagai default untuk`RetryMode`. Default untuk `MaxErrorRetry` dikonfigurasi dengan nilai yang sesuai dari tabel

sebelumnya kecuali nilai untuk kesalahan maksimum juga telah ditetapkan di lingkungan Anda (dijelaskan selanjutnya).

- Jika nilai untuk kesalahan maksimum telah ditetapkan di lingkungan Anda, nilai tersebut digunakan sebagai default untuk`MaxErrorRetry`. Amazon DynamoDB adalah pengecualian untuk aturan ini; nilai default DynamoDB `MaxErrorRetry` untuk selalu nilai dari tabel sebelumnya.

Saat aplikasi Anda berjalan

Saat membuat klien layanan, Anda dapat menggunakan nilai default untuk `RetryMode` dan`MaxErrorRetry`, seperti yang dijelaskan sebelumnya, atau Anda dapat menentukan nilai lainnya. Untuk menentukan nilai lain, buat dan sertakan objek konfigurasi layanan seperti AmazonDynamoDBConfigatau Amazon SQSConfig saat Anda membuat klien layanan.

Nilai-nilai ini tidak dapat diubah untuk klien layanan setelah dibuat.

Pertimbangan-pertimbangan

Ketika percobaan ulang terjadi, latensi permintaan Anda meningkat. Anda harus mengonfigurasi percobaan ulang berdasarkan batas aplikasi untuk latensi permintaan total dan tingkat kesalahan.

Timeout

AWS SDK untuk .NET Ini memungkinkan Anda untuk mengonfigurasi batas waktu permintaan di tingkat klien layanan dan per panggilan metode. Ada dua mekanisme untuk mengonfigurasi batas waktu, yang tercakup dalam bagian selanjutnya:

- Jika Anda menggunakan panggilan asinkron, Anda dapat menggunakan `CancellationToken` parameter metode ini.
- Jika Anda menggunakan panggilan sinkron di.NET Framework, Anda dapat menggunakan `Timeout` dan `ReadWriteTimeout` properti Amazon.Runtime abstrak. ClientConfigkelas.

Menggunakan **`CancellationToken`** parameter untuk batas waktu

AWS SDK untuk .NET Ini memungkinkan Anda untuk mengonfigurasi batas waktu permintaan pada panggilan asinkron dengan menggunakan parameter. `CancellationToken` Cuplikan kode berikut menunjukkan contoh. Kode melempar `System.Threading.Tasks.TaskCanceledException` jika permintaan tidak selesai dalam 10 detik.

```
string bucketName = "amzn-s3-demo-bucket";
string path = "pathToBucket";
using (var amazonS3Client = new AmazonS3Client(new AmazonS3Config()))
{
    // Cancel request after 10 seconds
    CancellationTokenSource cancellationTokenSource = new
 CancellationTokenSource(TimeSpan.FromMilliseconds(10000));
    CancellationToken cancellationToken = cancellationTokenSource.Token;
    ListObjectsV2Request listRequestV2 = new()
    {
        BucketName = bucketName,
        Prefix = path,
    };

    ListObjectsV2Response listResponseV2 = await
 amazonS3Client.ListObjectsV2Async(listRequestV2, cancellationToken);
}
```

Menggunakan **`Timeout`** dan **`ReadWriteTimeout`** properti untuk batas waktu

> **Note**
>
> `Timeout`Properti tidak memengaruhi panggilan asinkron. Jika Anda menggunakan panggilan asinkron, lihat sebagai gantinya. Menggunakan `CancellationToken` parameter untuk batas waktu

AWS SDK untuk .NET Ini memungkinkan Anda untuk mengonfigurasi batas waktu permintaan dan nilai read/write batas waktu soket di tingkat klien layanan. Nilai-nilai ini ditentukan dalam `Timeout` dan `ReadWriteTimeout` properti abstrak Amazon.Runtime. ClientConfigkelas. Nilai-nilai ini diteruskan sebagai `Timeout` dan `ReadWriteTimeout` properti dari HttpWebRequestobjek yang dibuat oleh objek klien AWS layanan. Secara default, `Timeout` nilainya 100 detik dan `ReadWriteTimeout` nilainya 300 detik.

Ketika jaringan Anda memiliki latensi tinggi, atau ada kondisi yang menyebabkan operasi dicoba ulang, menggunakan nilai batas waktu yang lama dan jumlah percobaan ulang yang tinggi dapat menyebabkan beberapa operasi SDK tampak tidak responsif.

> **Note**
>
> Versi AWS SDK untuk .NET yang menargetkan perpustakaan kelas portabel (PCL) menggunakan HttpClientkelas bukan `HttpWebRequest` kelas, dan hanya mendukung properti Timeout.

Berikut ini adalah pengecualian untuk nilai batas waktu default. Nilai-nilai ini diganti ketika Anda secara eksplisit menetapkan nilai batas waktu.

- `Timeout`dan `ReadWriteTimeout` disetel ke nilai maksimum jika metode yang dipanggil mengunggah aliran, seperti Amazons3Client. PutObjectAsync(), Amazons3Client. UploadPartAsync(), AmazonGlacierClient. UploadArchiveAsync(), dan sebagainya.
- Versi dari target AWS SDK untuk .NET yang ditetapkan .NET Framework `Timeout` dan `ReadWriteTimeout` ke nilai maksimum untuk semua Amazons3Client dan objek. AmazonGlacierClient
- Versi AWS SDK untuk .NET yang menargetkan perpustakaan kelas portabel (PCL) dan .NET Core diatur `Timeout` ke nilai maksimum untuk semua Amazons3Client dan objek. AmazonGlacierClient

Contoh berikut menunjukkan cara menentukan mode coba ulang Standar, maksimal 3 percobaan ulang, batas waktu 10 detik, dan batas read/write waktu 10 detik (jika ada). Konstruktor Amazons3Client diberikan objek Amazons3config.

```
var s3Client = new AmazonS3Client(
  new AmazonS3Config
  {
    Timeout = TimeSpan.FromSeconds(10),
    // NOTE: The following property is obsolete for
    //       versions of the AWS SDK untuk .NET that target .NET Core.
    ReadWriteTimeout = TimeSpan.FromSeconds(10),
    RetryMode = RequestRetryMode.Standard,
    MaxErrorRetry = 3
  });
```

Observabilitas

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

AWS SDK untuk .NET Dapat menyediakan dua sinyal telemetri umum, metrik dan jejak, serta pencatatan. Anda dapat menghubungkan data telemetri TelemetryProvideruntuk mengirim ke backend observabilitas (seperti atau AWS X-RayAmazon CloudWatch) dan kemudian menindaklanjutinya.

Secara default, sinyal telemetri dinonaktifkan di SDK. Topik ini menjelaskan cara mengaktifkan dan mengonfigurasi keluaran telemetri.

Sumber daya tambahan

Untuk informasi selengkapnya tentang mengaktifkan dan menggunakan observabilitas, lihat sumber daya berikut:

- OpenTelemetry
- Posting blog Enhancing Observability in the with. AWS SDK untuk .NET OpenTelemetry
- Posting blog Mengumumkan ketersediaan umum OpenTelemetry pustaka AWS .NET.
- Eksportir untuk OpenTelemetry
- Untuk contoh observabilitas di Alat AWS untuk PowerShell, lihat Observabilitas di Alat untuk PowerShell Panduan Pengguna.

Konfigurasikan `TelemetryProvider`

Anda dapat mengonfigurasi a `TelemetryProvider` dalam aplikasi Anda secara global untuk semua klien layanan atau untuk klien individu, seperti yang ditunjukkan dalam contoh berikut. the section called “Penyedia telemetri”Bagian ini berisi informasi tentang implementasi telemetri, termasuk informasi tentang implementasi yang disediakan bersama SDK.

Konfigurasikan penyedia telemetri global default

Secara default, setiap klien layanan mencoba menggunakan penyedia telemetri yang tersedia secara global. Dengan cara ini, Anda dapat mengatur penyedia sekali, dan semua klien akan menggunakannya. Ini harus dilakukan hanya sekali, sebelum Anda membuat klien layanan apa pun.

Cuplikan kode berikut menunjukkan cara mengatur penyedia telemetri global. Kemudian menciptakan klien layanan Amazon S3 dan mencoba melakukan operasi yang gagal. Kode menambahkan penelusuran dan metrik ke aplikasi. Kode ini menggunakan NuGet paket-paket berikut: `OpenTelemetry.Exporter.Console` dan`OpenTelemetry.Instrumentation.AWS`.

> **Note**
>
> Jika Anda menggunakan AWS IAM Identity Center untuk otentikasi, pastikan untuk juga menambahkan `AWSSDK.SSO` dan`AWSSDK.SSOOIDC`.

```
using Amazon.S3;
using OpenTelemetry;
using OpenTelemetry.Metrics;
using OpenTelemetry.Resources;
using OpenTelemetry.Trace;

Sdk.CreateTracerProviderBuilder()
    .ConfigureResource(e => e.AddService("DemoOtel"))
    .AddAWSInstrumentation()
    .AddConsoleExporter()
    .Build();

Sdk.CreateMeterProviderBuilder()
    .ConfigureResource(e => e.AddService("DemoOtel"))
    .AddAWSInstrumentation()
    .AddConsoleExporter()
    .Build();

var s3Client = new AmazonS3Client();

try
{
    var listBucketsResponse = await s3Client.ListBucketsAsync();
    // Attempt to delete a bucket that doesn't exist.
    var deleteBucketResponse = await s3Client.DeleteBucketAsync("amzn-s3-demo-bucket");
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}

Console.Read();
```

Konfigurasikan penyedia telemetri untuk klien layanan tertentu

Anda dapat mengonfigurasi klien layanan individual dengan penyedia telemetri tertentu (selain yang global). Untuk melakukannya, gunakan `TelemetryProvider` kelas objek Config dari konstruktor klien layanan. Misalnya, lihat Amazons3config dan cari propertinya. `TelemetryProvider` Lihat the section called “Penyedia telemetri” untuk informasi tentang implementasi telemetri kustom.

Topik

- Metrik
- Penyedia telemetri

Metrik

Tabel berikut mencantumkan metrik telemetri yang dipancarkan SDK. Konfigurasikan penyedia telemetri untuk membuat metrik dapat diamati.

Metrik apa yang dipancarkan?

Nama metrik	Unit	Tipe	Atribut	Deskripsi
client.call.durati on	detik	Histogra	rpc.service, rpc.method	Durasi panggilan keseluruhan (termasuk percobaan ulang dan waktu untuk mengirim atau menerima permintaan dan badan respons)
client.uptime	detik	Histogra	rpc.layanan	Jumlah waktu sejak klien dibuat
client.call.attemp ts	{mencob	Monoton Counter	rpc.service, rpc.method	Jumlah upaya untuk operasi individu
client.call.errors	{kesalah n}	Monoton Counter	rpc.service, rpc.method, exception.type	Jumlah kesalahan untuk suatu operasi
client.call.attemp t_duration	detik	Histogra	rpc.service, rpc.method	Waktu yang diperlukan untuk terhubung ke layanan, mengirim permintaan, dan mendapatkan kembali kode status HTTP dan

Nama metrik	Unit	Tipe	Atribut	Deskripsi
				header (termasuk waktu antrian menunggu untuk dikirim)
client.call.resolve_endpoint_duration	detik	Histogra	rpc.service, rpc.method	Waktu yang dibutuhkan untuk menyelesaikan titik akhir (endpoint resolver, bukan DNS) untuk permintaan
client.call.serialization_duration	detik	Histogra	rpc.service, rpc.method	Waktu yang dibutuhkan untuk membuat serial badan pesan
client.call.deserialization_duration	detik	Histogra	rpc.service, rpc.method	Waktu yang dibutuhkan untuk deserialisasi badan pesan
client.call.auth.signing_duration	detik	Histogra	rpc.service, rpc.method	Waktu yang dibutuhkan untuk menandatangani permintaan
client.call.auth.resolve_identity_duration	detik	Histogra	rpc.service, rpc.method	Waktu yang dibutuhkan untuk memperoleh identitas (seperti AWS kredensial atau token pembawa) dari Penyedia Identitas
client.http.bytes_sent	Oleh	Monoton Counter	server.address	Jumlah total byte yang dikirim oleh klien HTTP
client.http.bytes_received	Oleh	Monoton Counter	server.address	Jumlah total byte yang diterima oleh klien HTTP

Berikut ini adalah deskripsi kolom:

- Nama metrik —Nama metrik yang dipancarkan.
- Satuan —Satuan ukuran untuk metrik. Unit diberikan dalam notasi UCUM case sensitive (“c/s”).
- Jenis —Jenis instrumen yang digunakan untuk menangkap metrik.
- Atribut —Himpunan atribut (dimensi) yang dipancarkan dengan metrik.
- Deskripsi —Deskripsi tentang apa yang diukur metrik.

Penyedia telemetri

SDK menyediakan implementasi OpenTelemetrysebagai penyedia telemetri, yang dijelaskan di bagian selanjutnya.

Jika Anda memiliki persyaratan telemetri tertentu, sudah memiliki solusi telemetri dalam pikiran, atau memerlukan kontrol halus atas bagaimana data telemetri ditangkap dan diproses, Anda juga dapat menerapkan penyedia telemetri Anda sendiri.

Daftarkan implementasi Anda sendiri dengan TelemetryProviderkelas. Berikut ini adalah contoh sederhana tentang cara mendaftar sendiri `TracerProvider` dan`MeterProvider`.

```
using Amazon;
using Amazon.Runtime.Telemetry;
using Amazon.Runtime.Telemetry.Metrics;
using Amazon.Runtime.Telemetry.Tracing;

public class CustomTracerProvider : TracerProvider
{
    // Implement custom tracing logic here
}
public class CustomMeterProvider : MeterProvider
{
    // Implement custom metrics logic here
}

// Register custom implementations
AWSConfigs.TelemetryProvider.RegisterTracerProvider(new CustomTracerProvider());
AWSConfigs.TelemetryProvider.RegisterMeterProvider(new CustomMeterProvider());
```

Topik

- Konfigurasikan penyedia telemetri OpenTelemetry berbasis

Konfigurasikan penyedia telemetri OpenTelemetry berbasis

AWS SDK untuk .NET Termasuk implementasi penyedia telemetri OpenTelemetry berbasis. Untuk detail tentang cara menetapkan penyedia ini sebagai penyedia telemetri global, lihat. Konfigurasikan `TelemetryProvider` Untuk menggunakan penyedia telemetri ini, Anda memerlukan sumber daya berikut dalam proyek Anda:

- Paket OpenTelemetry NuGet .instrumentation.aws.
- Eksportir telemetri seperti OTLP atau Console. Untuk informasi selengkapnya, lihat Eksportir dalam OpenTelemetry dokumentasi.

OpenTelemetry Implementasi yang disertakan dengan SDK dapat dikonfigurasi untuk mengurangi jumlah penelusuran untuk permintaan, kredensi, dan kompresi HTTPS. Untuk melakukannya, atur `SuppressDownstreamInstrumentation` opsi ke`true`, mirip dengan yang berikut ini:

```
Sdk.CreateTracerProviderBuilder()
    .ConfigureResource(e => e.AddService("DemoOtel"))
    .AddAWSInstrumentation(options => options.SuppressDownstreamInstrumentation = true)
    .AddConsoleExporter()
    .Build();
```

Untuk informasi tambahan tentang penyedia ini, lihat posting blog Enhancing Observability in the with. AWS SDK untuk .NET OpenTelemetry

Informasi tambahan tentang pengguna dan peran

Untuk melakukan pengembangan.NET pada AWS atau untuk menjalankan aplikasi.NET AWS, Anda perlu memiliki beberapa kombinasi pengguna, set izin, dan peran layanan yang sesuai untuk tugas-tugas ini.

Pengguna tertentu, set izin, dan peran layanan yang Anda buat, dan cara Anda menggunakannya, akan bergantung pada persyaratan aplikasi Anda. Berikut ini adalah beberapa informasi tambahan tentang mengapa mereka dapat digunakan dan cara membuatnya.

Pengguna dan set izin

Meskipun dimungkinkan untuk menggunakan akun pengguna IAM dengan kredensi jangka panjang untuk mengakses AWS layanan, ini bukan lagi praktik terbaik dan harus dihindari. Bahkan selama pengembangan, itu adalah praktik terbaik untuk membuat pengguna dan set izin AWS IAM Identity Center dan menggunakan kredensi sementara yang disediakan oleh sumber identitas.

Untuk pengembangan, Anda dapat menggunakan pengguna yang Anda buat atau diberikanAutentikasi dengan AWS. Jika Anda memiliki Konsol Manajemen AWS izin yang sesuai, Anda juga dapat membuat set izin yang berbeda dengan hak istimewa paling sedikit untuk pengguna tersebut atau membuat pengguna baru khusus untuk proyek pengembangan, memberikan set izin

dengan hak istimewa paling sedikit. Tindakan yang Anda pilih, jika ada, tergantung pada keadaan Anda.

Untuk informasi selengkapnya tentang pengguna dan set izin ini serta cara membuatnya, lihat Otentikasi dan akses di Panduan Referensi Alat AWS SDKs dan Memulai di Panduan AWS IAM Identity Center Pengguna.

Peran layanan

Anda dapat mengatur peran AWS layanan untuk mengakses AWS layanan atas nama pengguna. Jenis akses ini sesuai jika beberapa orang akan menjalankan aplikasi Anda dari jarak jauh; misalnya, pada EC2 instance Amazon yang telah Anda buat untuk tujuan ini.

Proses untuk membuat peran layanan bervariasi tergantung pada situasinya, tetapi pada dasarnya adalah sebagai berikut.

1. Masuk ke Konsol Manajemen AWS dan buka konsol IAM di https://console.aws.amazon.com/iam/.
2. Pilih Peran, lalu pilih Buat peran.
3. Pilih AWS layanan, temukan dan pilih EC2(misalnya), lalu pilih kasus EC2penggunaan (misalnya).
4. Pilih Berikutnya: Izin, dan pilih kebijakan yang sesuai untuk AWS layanan yang akan digunakan aplikasi Anda.

   > ⚠ Warning
   >
   > JANGAN memilih AdministratorAccesskebijakan karena kebijakan tersebut memungkinkan izin baca dan tulis untuk hampir semua yang ada di akun Anda.

5. Pilih Berikutnya: Tag dan masukkan tag apa pun yang Anda inginkan.

   Anda dapat menemukan informasi tentang tag di Akses kontrol menggunakan tag AWS sumber daya di Panduan Pengguna IAM.
6. Pilih Berikutnya: Tinjau dan berikan nama Peran dan deskripsi Peran. Lalu pilih Buat peran.

Anda dapat menemukan informasi tingkat tinggi tentang peran IAM di Identitas (pengguna, grup, dan peran) di Panduan Pengguna IAM. Temukan informasi terperinci tentang peran dalam topik peran IAM panduan itu.

Informasi tambahan tentang peran

- Gunakan peran IAM untuk tugas untuk tugas Amazon Elastic Container Service (Amazon ECS).
- Gunakan peran IAM untuk aplikasi yang berjalan di EC2 instans Amazon.

Konfigurasi lanjutan untuk AWS SDK untuk .NET proyek Anda

Topik di bagian ini berisi informasi tentang tugas dan metode konfigurasi tambahan yang mungkin menarik bagi Anda.

Topik

- Menggunakan AWSSDK .Extensions. NETCore.Setup dan antarmuka IConfiguration
- Mengkonfigurasi Parameter Aplikasi Lainnya
- Referensi File Konfigurasi untuk AWS SDK untuk .NET

Menggunakan AWSSDK .Extensions. NETCore.Setup dan antarmuka IConfiguration

(Topik ini sebelumnya berjudul, “Configuring the AWS SDK untuk .NET with .NET Core”)

Salah satu perubahan terbesar dalam .NET Core adalah penghapusan dan standar `ConfigurationManager` `app.config` dan `web.config` file yang digunakan dengan aplikasi.NET Framework dan ASP.NET.

Konfigurasi dalam.NET Core didasarkan pada pasangan nilai kunci yang ditetapkan oleh penyedia konfigurasi. Penyedia konfigurasi membaca data konfigurasi menjadi pasangan nilai kunci dari berbagai sumber konfigurasi, termasuk argumen baris perintah, file direktori, variabel lingkungan, dan file pengaturan.

> **Note**
>
> Untuk informasi lebih lanjut, lihat Konfigurasi di ASP.NET Core.

Untuk membuatnya mudah untuk menggunakan AWS SDK untuk .NET dengan .NET Core, Anda dapat menggunakan AWSSDK.Extensions. NETCore.Setup NuGet paket. Seperti banyak

pustaka .NET Core, ia menambahkan metode ekstensi ke `IConfiguration` antarmuka untuk membuat AWS konfigurasi mulus.

Kode sumber untuk paket ini aktif GitHub dihttps://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.NETCore.Setup.

Menggunakan AWSSDK .Extensions. NETCore.Pengaturan

Misalkan Anda membuat aplikasi ASP.NET Core Model-View-Controller (MVC), yang dapat dicapai dengan template Aplikasi Web ASP.NET Core di Visual Studio atau dengan berjalan `dotnet new mvc ...` di .NET Core CLI. Ketika Anda membuat aplikasi seperti itu, konstruktor untuk `Startup.cs` menangani konfigurasi dengan membaca di berbagai sumber input dari penyedia konfigurasi sepertiappsettings.json.

```
public Startup(IConfiguration configuration)
{
    Configuration = configuration;
}
```

Untuk menggunakan `Configuration` objek untuk mendapatkan AWSopsi, pertama-tama tambahkan `AWSSDK.Extensions.NETCore.Setup` NuGet paket. Kemudian, tambahkan opsi Anda ke file konfigurasi seperti yang dijelaskan selanjutnya.

Perhatikan bahwa salah satu file yang ditambahkan ke proyek Anda adalahappsettings.Development.json. Ini sesuai dengan satu `EnvironmentName` set untuk Pengembangan. Selama pengembangan, Anda menempatkan konfigurasi Anda dalam file ini, yang hanya dibaca selama pengujian lokal. Saat Anda menerapkan instans Amazon EC2 yang `EnvironmentName` telah disetel ke Production, file ini akan diabaikan dan AWS SDK untuk .NET akan kembali ke kredensi IAM dan Wilayah yang dikonfigurasi untuk instans Amazon EC2.

Pengaturan konfigurasi berikut menunjukkan contoh nilai yang dapat Anda tambahkan dalam `appsettings.Development.json` file dalam proyek Anda untuk menyediakan AWS pengaturan.

```
{
  "AWS": {
    "Profile": "local-test-profile",
    "Region": "us-west-2"
  },
  "SupportEmail": "TechSupport@example.com"
}
```

Untuk mengakses pengaturan dalam file CSHTHTML, gunakan direktif. `Configuration`

```
@using Microsoft.Extensions.Configuration
@inject IConfiguration Configuration

<h1>Contact</h1>

<p>
    <strong>Support:</strong> <a
 href='mailto:@Configuration["SupportEmail"]'>@Configuration["SupportEmail"]</a><br />
</p>
```

Untuk mengakses AWS opsi yang ditetapkan dalam file dari kode, panggil metode `GetAWSOptions` ekstensi yang ditambahkan ke`IConfiguration`.

Untuk membangun klien layanan dari opsi ini, hubungi`CreateServiceClient`. Contoh berikut menunjukkan cara membuat klien layanan Amazon S3. (Pastikan untuk menambahkan AWSSDK NuGet paket.S3 ke proyek Anda.)

```
var options = Configuration.GetAWSOptions();
IAmazonS3 client = options.CreateServiceClient<IAmazonS3>();
```

Anda juga dapat membuat beberapa klien layanan dengan pengaturan yang tidak kompatibel dengan menggunakan beberapa entri dalam *`appsettings.Development.json`* file, seperti yang ditunjukkan dalam contoh berikut di mana konfigurasi untuk *`service1`* menyertakan *`us-west-2`* Wilayah dan konfigurasi untuk *`service2`* menyertakan URL titik akhir khusus.

```
{
  "service1": {
    "Profile": "default",
    "Region": "us-west-2"
  },
  "service2": {
    "Profile": "default",
    "ServiceURL": "URL"
  }
}
```

Anda kemudian bisa mendapatkan opsi untuk layanan tertentu dengan menggunakan entri dalam file JSON. Misalnya, untuk mendapatkan pengaturan untuk `service1` gunakan yang berikut ini.

```
var options = Configuration.GetAWSOptions("service1");
```

Nilai yang diizinkan dalam file pengaturan aplikasi

Nilai konfigurasi aplikasi berikut dapat diatur dalam `appsettings.Development.json` file. Nama bidang harus menggunakan casing yang ditampilkan. Untuk detail tentang pengaturan ini, lihat `AWS.Runtime.ClientConfig` kelas.

- Region
- Profil
- ProfilesLocation
- SignatureVersion
- RegionEndpoint
- UseHttp
- ServiceURL
- AuthenticationRegion
- AuthenticationServiceName
- MaxErrorRetry
- LogResponse
- BufferSize
- ProgressUpdateInterval
- ResignRetries
- AllowAutoRedirect
- LogMetrics
- DisableLogging
- UseDualstackEndpoint

Injeksi ketergantungan ASP.NET Core

AWSSDK.Extensions. NETCore NuGet Paket .Setup juga terintegrasi dengan sistem injeksi ketergantungan baru di ASP.NET Core. `ConfigureServices`Metode di `Startup` kelas aplikasi Anda adalah di mana layanan MVC ditambahkan. Jika aplikasi menggunakan Entity Framework, ini juga tempat yang diinisialisasi.

```
public void ConfigureServices(IServiceCollection services)
{
    // Add framework services.
    services.AddMvc();
}
```

> **Note**
>
> Latar belakang injeksi ketergantungan di.NET Core tersedia di situs dokumentasi.NET Core.

`AWSSDK.Extensions.NETCore.Setup` NuGet Paket menambahkan metode ekstensi baru `IServiceCollection` yang dapat Anda gunakan untuk menambahkan AWS layanan ke injeksi ketergantungan. Kode berikut menunjukkan cara menambahkan AWS opsi yang dibaca `IConfiguration` untuk menambahkan Amazon S3 dan DynamoDB ke daftar layanan. (Pastikan untuk menambahkan AWSSDKDBv2 NuGet paket.S3 dan AWSSDK.Dynamo ke proyek Anda.)

```
public void ConfigureServices(IServiceCollection services)
{
    // Add framework services.
    services.AddMvc();

    services.AddDefaultAWSOptions(Configuration.GetAWSOptions());
    services.AddAWSService<IAmazonS3>();
    services.AddAWSService<IAmazonDynamoDB>();
}
```

Sekarang, jika pengontrol MVC Anda menggunakan salah satu `IAmazonS3` atau `IAmazonDynamoDB` sebagai parameter dalam konstruktornya, sistem injeksi ketergantungan melewati layanan tersebut.

```
public class HomeController : Controller
{
    IAmazonS3 S3Client { get; set; }

    public HomeController(IAmazonS3 s3Client)
    {
        this.S3Client = s3Client;
    }
```

```
    ...

}
```

Mengkonfigurasi Parameter Aplikasi Lainnya

> **Note**
>
> Informasi dalam topik ini khusus untuk proyek berdasarkan .NET Framework. `Web.config`File `App.config` dan tidak ada secara default dalam proyek berdasarkan .NET Core.

Terbuka untuk melihat konten.NET Framework

Ada sejumlah parameter aplikasi yang dapat Anda konfigurasikan:

- `AWSLogging`
- `AWSLogMetrics`
- `AWSRegion`
- `AWSResponseLogging`
- `AWS.DynamoDBContext.TableNamePrefix`
- `AWS.S3.UseSignatureVersion4`
- `AWSEndpointDefinition`
- AWS Titik Akhir yang Dihasilkan Layanan

Parameter ini dapat dikonfigurasi dalam aplikasi `App.config` atau `Web.config` file. Meskipun Anda juga dapat mengonfigurasinya dengan AWS SDK untuk .NET API, kami sarankan Anda menggunakan `.config` file aplikasi. Kedua pendekatan tersebut dijelaskan di sini.

Untuk informasi selengkapnya tentang penggunaan `<aws>` elemen seperti yang dijelaskan nanti dalam topik ini, lihat Referensi File Konfigurasi untuk AWS SDK untuk .NET.

AWSLogging

Mengonfigurasi bagaimana SDK harus mencatat peristiwa, jika ada. Misalnya, pendekatan yang disarankan adalah dengan menggunakan `<logging>` elemen, yang merupakan elemen anak dari `<aws>` elemen:

```
<aws>
  <logging logTo="Log4Net"/>
</aws>
```

Atau:

```
<add key="AWSLogging" value="log4net"/>
```

Nilai yang mungkin adalah:

`None`

Matikan pencatatan peristiwa. Ini adalah opsi default.

`log4net`

Log menggunakan log4net.

`SystemDiagnostics`

Log menggunakan `System.Diagnostics` kelas.

Anda dapat mengatur beberapa nilai untuk `logTo` atribut, dipisahkan dengan koma. Contoh berikut menetapkan keduanya `log4net` dan `System.Diagnostics` masuk ke dalam `.config` file:

```
<logging logTo="Log4Net, SystemDiagnostics"/>
```

Atau:

```
<add key="AWSLogging" value="log4net, SystemDiagnostics"/>
```

Atau, dengan menggunakan AWS SDK untuk .NET API, gabungkan nilai LoggingOptionsenumerasi dan setel properti .Logging: AWSConfigs

```
AWSConfigs.Logging = LoggingOptions.Log4Net | LoggingOptions.SystemDiagnostics;
```

Perubahan pada pengaturan ini hanya berlaku untuk instance AWS klien baru.

AWSLogMetrik

Menentukan apakah SDK harus mencatat metrik kinerja atau tidak. Untuk mengatur konfigurasi logging metrik dalam `.config` file, atur nilai `logMetrics` atribut dalam `<logging>` elemen, yang merupakan elemen anak dari `<aws>` elemen:

```
<aws>
  <logging logMetrics="true"/>
</aws>
```

Atau, atur `AWSLogMetrics` kunci di `<appSettings>` bagian:

```
<add key="AWSLogMetrics" value="true">
```

Atau, untuk menyetel pencatatan metrik dengan AWS SDK untuk .NET API, setel file. AWSConfigs LogMetricsproperti:

```
AWSConfigs.LogMetrics = true;
```

Pengaturan ini mengkonfigurasi `LogMetrics` properti default untuk semua klien/konfigurasi. Perubahan pada pengaturan ini hanya berlaku untuk instance AWS klien baru.

AWSRegion

Mengkonfigurasi AWS wilayah default untuk klien yang belum secara eksplisit menentukan wilayah. Untuk mengatur wilayah dalam `.config` file, pendekatan yang disarankan adalah mengatur nilai `region` atribut dalam `aws` elemen:

```
<aws region="us-west-2"/>
```

Atau, atur AWSRegionkunci di `<appSettings>` bagian:

```
<add key="AWSRegion" value="us-west-2"/>
```

Atau, untuk menyetel wilayah dengan AWS SDK untuk .NET API, atur AWSConfigs. AWSRegionproperti:

```
AWSConfigs.AWSRegion = "us-west-2";
```

Untuk informasi selengkapnya tentang membuat AWS klien untuk wilayah tertentu, lihat Pemilihan AWS Wilayah. Perubahan pada pengaturan ini hanya berlaku untuk instance AWS klien baru.

AWSResponsePenebangan

Mengkonfigurasi kapan SDK harus mencatat respons layanan. Nilai yang mungkin adalah:

`Never`

Jangan pernah mencatat tanggapan layanan. Ini adalah opsi default.

`Always`

Selalu log tanggapan layanan.

`OnError`

Hanya log tanggapan layanan ketika terjadi kesalahan.

Untuk mengatur konfigurasi pencatatan layanan dalam `.config` file, pendekatan yang disarankan adalah mengatur nilai `logResponses` atribut dalam `<logging>` elemen, yang merupakan elemen anak dari `<aws>` elemen:

```
<aws>
  <logging logResponses="OnError"/>
</aws>
```

Atau, atur kunci AWSResponseLogging di `<appSettings>` bagian:

```
<add key="AWSResponseLogging" value="OnError"/>
```

Atau, untuk mengatur pencatatan layanan dengan AWS SDK untuk .NET API, setel file AWSConfigs.ResponseLoggingproperti ke salah satu nilai ResponseLoggingOptionpencacahan:

```
AWSConfigs.ResponseLogging = ResponseLoggingOption.OnError;
```

Perubahan pada pengaturan ini segera berlaku.

`AWS.DynamoDBContext.TableNamePrefix`

Mengkonfigurasi default `TableNamePrefix` yang `DynamoDBContext` akan digunakan jika tidak dikonfigurasi secara manual.

Untuk mengatur awalan nama tabel dalam `.config` file, pendekatan yang disarankan adalah mengatur nilai `tableNamePrefix` atribut dalam `<dynamoDBContext>` elemen, yang merupakan elemen anak dari `<dynamoDB>` elemen, yang merupakan elemen anak dari `<aws>` elemen:

```
<dynamoDBContext tableNamePrefix="Test-"/>
```

Atau, atur `AWS.DynamoDBContext.TableNamePrefix` kunci di `<appSettings>` bagian:

```
<add key="AWS.DynamoDBContext.TableNamePrefix" value="Test-"/>
```

Atau, untuk mengatur awalan nama tabel dengan AWS SDK untuk .NET API, setel properti AWSConfigs.Dynamo DBContext TableNamePrefix:

```
AWSConfigs.DynamoDBContextTableNamePrefix = "Test-";
```

Perubahan pada pengaturan ini hanya akan berlaku dalam contoh yang baru dibangun dari `DynamoDBContextConfig` dan`DynamoDBContext`.

`AWS.S3.UseSignatureVersion4`

Mengkonfigurasi apakah klien Amazon S3 harus menggunakan tanda tangan versi 4 penandatanganan dengan permintaan atau tidak.

Untuk mengatur tanda tangan versi 4 penandatanganan untuk Amazon S3 dalam `.config` file, pendekatan yang disarankan adalah mengatur `useSignatureVersion4` atribut `<s3>` elemen, yang merupakan elemen turunan dari elemen: `<aws>`

```
<aws>
  <s3 useSignatureVersion4="true"/>
</aws>
```

Atau, atur `AWS.S3.UseSignatureVersion4` kuncinya `true` di `<appSettings>` bagian:

```
<add key="AWS.S3.UseSignatureVersion4" value="true"/>
```

Atau, untuk menyetel tanda tangan penandatanganan versi 4 dengan AWS SDK untuk .NET API, setel AWSConfigsproperti.S3 UseSignatureVersion 4 ke: `true`

```
AWSConfigs.S3UseSignatureVersion4 = true;
```

Secara default, pengaturan ini`false`, tetapi versi tanda tangan 4 dapat digunakan secara default dalam beberapa kasus atau dengan beberapa wilayah. Saat pengaturannya`true`, tanda tangan versi 4 akan digunakan untuk semua permintaan. Perubahan pada setelan ini hanya berlaku untuk instans klien Amazon S3 baru.

AWSEndpointDefinisi

Mengkonfigurasi apakah SDK harus menggunakan file konfigurasi khusus yang mendefinisikan wilayah dan titik akhir.

Untuk mengatur file definisi titik akhir dalam `.config` file, kami sarankan untuk mengatur nilai `endpointDefinition` atribut dalam `<aws>` elemen.

```
<aws endpointDefinition="c:\config\endpoints.json"/>
```

Atau, Anda dapat mengatur kunci AWSEndpointDefinisi di `<appSettings>` bagian:

```
<add key="AWSEndpointDefinition" value="c:\config\endpoints.json"/>
```

Atau, untuk menyetel file definisi titik akhir dengan AWS SDK untuk .NET API, setel file. AWSConfigs EndpointDefinitionproperti:

```
AWSConfigs.EndpointDefinition = @"c:\config\endpoints.json";
```

Jika tidak ada nama file yang disediakan, maka file konfigurasi khusus tidak akan digunakan. Perubahan pada pengaturan ini hanya berlaku untuk instance AWS klien baru. `endpoints.json`File tersedia darihttps://github.com/aws/aws-sdk-net/blob/main/sdk/src/Core/endpoints.json.

AWS Titik Akhir yang Dihasilkan Layanan

Beberapa AWS layanan menghasilkan titik akhir mereka sendiri alih-alih menggunakan titik akhir wilayah. Klien untuk layanan ini menggunakan URL layanan yang khusus untuk layanan itu dan sumber daya Anda. Dua contoh layanan ini adalah Amazon CloudSearch dan AWS IoT. Contoh berikut menunjukkan bagaimana Anda dapat memperoleh titik akhir untuk layanan tersebut.

Contoh CloudSearch Titik Akhir Amazon

CloudSearch Klien Amazon digunakan untuk mengakses layanan CloudSearch konfigurasi Amazon. Anda menggunakan layanan CloudSearch konfigurasi Amazon untuk membuat, mengonfigurasi, dan mengelola domain penelusuran. Untuk membuat domain

pencarian, buat CreateDomainRequestobjek dan berikan `DomainName` properti. Buat AmazonCloudSearchClientobjek dengan menggunakan objek permintaan. Panggil metode CreateDomain. CreateDomainResponseObjek yang dikembalikan dari panggilan berisi `DomainStatus` properti yang memiliki titik akhir `DocService` dan `SearchService` titik akhir. Buat AmazonCloudSearchDomainConfigobjek dan gunakan untuk menginisialisasi `DocService` dan `SearchService` contoh kelas. AmazonCloudSearchDomainClient

```
// Create domain and retrieve DocService and SearchService endpoints
DomainStatus domainStatus;
using (var searchClient = new AmazonCloudSearchClient())
{
    var request = new CreateDomainRequest
    {
        DomainName = "testdomain"
    };
    domainStatus = searchClient.CreateDomain(request).DomainStatus;
    Console.WriteLine(domainStatus.DomainName + " created");
}

// Test the DocService endpoint
var docServiceConfig = new AmazonCloudSearchDomainConfig
{
    ServiceURL = "https://" + domainStatus.DocService.Endpoint
};
using (var domainDocService = new AmazonCloudSearchDomainClient(docServiceConfig))
{
    Console.WriteLine("Amazon CloudSearchDomain DocService client instantiated using
 the DocService endpoint");
    Console.WriteLine("DocService endpoint = " + domainStatus.DocService.Endpoint);

    using (var docStream = new FileStream(@"C:\doc_source\XMLFile4.xml",
 FileMode.Open))
    {
        var upload = new UploadDocumentsRequest
        {
            ContentType = ContentType.ApplicationXml,
            Documents = docStream
        };
        domainDocService.UploadDocuments(upload);
    }
}

// Test the SearchService endpoint
```

```
var searchServiceConfig = new AmazonCloudSearchDomainConfig
{
    ServiceURL = "https://" + domainStatus.SearchService.Endpoint
};
using (var domainSearchService = new
 AmazonCloudSearchDomainClient(searchServiceConfig))
{
    Console.WriteLine("Amazon CloudSearchDomain SearchService client instantiated using
 the SearchService endpoint");
    Console.WriteLine("SearchService endpoint = " +
 domainStatus.SearchService.Endpoint);

    var searchReq = new SearchRequest
    {
        Query = "Gambardella",
        Sort = "_score desc",
        QueryParser = QueryParser.Simple
    };
    var searchResp = domainSearchService.Search(searchReq);
}
```

AWS IoT Contoh Titik Akhir

Untuk mendapatkan titik akhir untuk AWS IoT, buat AmazonIoTClientobjek dan panggil DescribeEndPointmetode. DescribeEndPointResponseObjek yang dikembalikan berisi file`EndpointAddress`. Buat AmazonIotDataConfigobjek, atur `ServiceURL` properti, dan gunakan objek untuk membuat instance kelas. AmazonIotDataClient

```
string iotEndpointAddress;
using (var iotClient = new AmazonIoTClient())
{
    var endPointResponse = iotClient.DescribeEndpoint();
    iotEndpointAddress = endPointResponse.EndpointAddress;
}

var ioTdocServiceConfig = new AmazonIotDataConfig
{
    ServiceURL = "https://" + iotEndpointAddress
};
using (var dataClient = new AmazonIotDataClient(ioTdocServiceConfig))
{
    Console.WriteLine("AWS IoTData client instantiated using the endpoint from the
 IotClient");
```

```
}
```

Referensi File Konfigurasi untuk AWS SDK untuk .NET

> **Note**
>
> Informasi dalam topik ini khusus untuk proyek berdasarkan .NET Framework. `Web.config`File `App.config` dan tidak ada secara default dalam proyek berdasarkan .NET Core.

Terbuka untuk melihat konten.NET Framework

Anda dapat menggunakan proyek `App.config` atau `Web.config` file .NET untuk menentukan AWS pengaturan, seperti AWS kredensional, opsi logging, titik akhir AWS layanan, dan AWS wilayah, serta beberapa pengaturan untuk AWS layanan, seperti Amazon DynamoDB, Amazon, dan Amazon S3. EC2 Informasi berikut menjelaskan cara memformat `Web.config` file `App.config` atau dengan benar untuk menentukan jenis pengaturan ini.

> **Note**
>
> Meskipun Anda dapat terus menggunakan `<appSettings>` elemen dalam `Web.config` file `App.config` atau untuk menentukan AWS pengaturan, kami sarankan Anda menggunakan `<aws>` elemen `<configSections>` dan seperti yang dijelaskan nanti dalam topik ini. Untuk informasi selengkapnya tentang `<appSettings>` elemen, lihat contoh `<appSettings>` elemen dalam Mengkonfigurasi AWS SDK untuk .NET Aplikasi Anda.

> **Note**
>
> Meskipun Anda dapat terus menggunakan properti `AWSConfigs`kelas berikut dalam file kode untuk menentukan AWS pengaturan, properti berikut tidak digunakan lagi dan mungkin tidak didukung dalam rilis mendatang:
>
> - `DynamoDBContextTableNamePrefix`
> - `EC2UseSignatureVersion4`
> - `LoggingOptions`

- `LogMetrics`
- `ResponseLoggingOption`
- `S3UseSignatureVersion4`

Secara umum, kami menyarankan bahwa alih-alih menggunakan properti `AWSConfigs` kelas dalam file kode untuk menentukan AWS pengaturan, Anda harus menggunakan `<aws>` elemen `<configSections>` dan dalam `Web.config` file `App.config` atau untuk menentukan AWS pengaturan, seperti yang dijelaskan nanti dalam topik ini. Untuk informasi selengkapnya tentang properti sebelumnya, lihat contoh `AWSConfigs` kode di Mengonfigurasi Aplikasi Anda. AWS SDK untuk .NET

Topik

- Mendeklarasikan Bagian AWS Pengaturan
- Elemen yang Diizinkan
- Referensi Elemen

Mendeklarasikan Bagian AWS Pengaturan

Anda menentukan AWS pengaturan dalam `Web.config` file `App.config` atau dari dalam `<aws>` elemen. Sebelum Anda dapat mulai menggunakan `<aws>` elemen, Anda harus membuat `<section>` elemen (yang merupakan elemen anak dari `<configSections>` elemen) dan mengatur `name` atributnya ke `aws` dan `type` atributnya`Amazon.AWSSection, AWSSDK.Core`, seperti yang ditunjukkan pada contoh berikut:

```
<?xml version="1.0"?>
<configuration>
  ...
  <configSections>
    <section name="aws" type="Amazon.AWSSection, AWSSDK.Core"/>
  </configSections>
  <aws>
    <!-- Add your desired AWS settings declarations here. -->
  </aws>
  ...
</configuration>
```

Visual Studio Editor tidak menyediakan penyelesaian kode otomatis (IntelliSense) untuk `<aws>` elemen atau elemen turunannya.

Untuk membantu Anda membuat versi `<aws>` elemen yang diformat dengan benar, panggil `Amazon.AWSConfigs.GenerateConfigTemplate` metode ini. Ini menghasilkan versi kanonik `<aws>` elemen sebagai string yang dicetak cantik, yang dapat Anda sesuaikan dengan kebutuhan Anda. Bagian berikut menjelaskan atribut `<aws>` elemen dan elemen anak.

Elemen yang Diizinkan

Berikut ini adalah daftar hubungan logis antara elemen yang diizinkan di bagian AWS pengaturan. Anda dapat menghasilkan versi terbaru dari daftar ini dengan memanggil `Amazon.AWSConfigs.GenerateConfigTemplate` metode, yang menghasilkan versi kanonik `<aws>` elemen sebagai string yang dapat Anda sesuaikan dengan kebutuhan Anda.

```
<aws
  endpointDefinition="string value"
  region="string value"
  profileName="string value"
  profilesLocation="string value">
  <logging
    logTo="None, Log4Net, SystemDiagnostics"
    logResponses="Never | OnError | Always"
    logMetrics="true | false"
    logMetricsFormat="Standard | JSON"
    logMetricsCustomFormatter="NameSpace.Class, Assembly" />
  <dynamoDB
    conversionSchema="V1 | V2">
    <dynamoDBContext
      tableNamePrefix="string value">
      <tableAliases>
        <alias
          fromTable="string value"
          toTable="string value" />
      </tableAliases>
      <map
        type="NameSpace.Class, Assembly"
        targetTable="string value">
        <property
          name="string value"
          attribute="string value"
          ignore="true | false"
```

```
          version="true | false"
          converter="NameSpace.Class, Assembly" />
      </map>
    </dynamoDBContext>
  </dynamoDB>
  <s3
    useSignatureVersion4="true | false" />
  <ec2
    useSignatureVersion4="true | false" />
  <proxy
    host="string value"
    port="1234"
    username="string value"
    password="string value" />
</aws>
```

Referensi Elemen

Berikut ini adalah daftar elemen yang diizinkan di bagian AWS pengaturan. Untuk setiap elemen, atribut yang diizinkan dan elemen induk-anak terdaftar.

Topik

- alias
- aws
- dynamoDB
- dinamo DBContext
- ec2
- pencatatan log
- map
- properti
- proxy
- s3

alias

`<alias>`Elemen mewakili satu item dalam koleksi satu atau lebih pemetaan dari-tabel ke tabel yang menentukan tabel yang berbeda dari yang dikonfigurasi untuk tipe. Elemen ini memetakan ke instance `Amazon.Util.TableAlias` kelas dari

`Amazon.AWSConfigs.DynamoDBConfig.Context.TableAliases` properti di AWS SDK untuk .NET. Pemetaan ulang dilakukan sebelum menerapkan awalan nama tabel.

Elemen ini dapat mencakup atribut berikut:

`fromTable`

Bagian dari tabel dari pemetaan dari tabel ke tabel ke tabel. Atribut ini memetakan ke `Amazon.Util.TableAlias.FromTable` properti di AWS SDK untuk .NET.

`toTable`

Bagian to-table dari pemetaan dari-tabel ke tabel ke tabel. Atribut ini memetakan ke `Amazon.Util.TableAlias.ToTable` properti di AWS SDK untuk .NET.

Induk `<alias>` elemen adalah `<tableAliases>` elemen.

`<alias>`Elemen tidak mengandung elemen anak.

Berikut ini adalah contoh `<alias>` elemen yang digunakan:

```
<alias
  fromTable="Studio"
  toTable="Studios" />
```

`aws`

`<aws>`Elemen mewakili elemen paling atas di bagian AWS pengaturan. Elemen ini dapat mencakup atribut berikut:

`endpointDefinition`

Jalur absolut ke file konfigurasi khusus yang mendefinisikan AWS wilayah dan titik akhir yang akan digunakan. Atribut ini memetakan ke `Amazon.AWSConfigs.EndpointDefinition` properti di AWS SDK untuk .NET.

`profileName`

Nama profil untuk AWS kredensi tersimpan yang akan digunakan untuk melakukan panggilan layanan. Atribut ini memetakan ke `Amazon.AWSConfigs.AWSProfileName` properti di AWS SDK untuk .NET.

`profilesLocation`

Jalur absolut ke lokasi file kredensial yang dibagikan dengan yang lain. AWS SDKs Secara default, file kredensial disimpan dalam `.aws` direktori di direktori home pengguna saat ini. Atribut ini memetakan ke `Amazon.AWSConfigs.AWSProfilesLocation` properti di AWS SDK untuk .NET.

`region`

ID AWS wilayah default untuk klien yang belum secara eksplisit menentukan wilayah. Atribut ini memetakan ke `Amazon.AWSConfigs.AWSRegion` properti di AWS SDK untuk .NET.

`<aws>`Elemen tidak memiliki elemen induk.

`<aws>`Elemen dapat mencakup elemen anak berikut:

- `<dynamoDB>`
- `<ec2>`
- `<logging>`
- `<proxy>`
- `<s3>`

Berikut ini adalah contoh `<aws>` elemen yang digunakan:

```
<aws
  endpointDefinition="C:\Configs\endpoints.xml"
  region="us-west-2"
  profileName="development"
  profilesLocation="C:\Configs">
  <!-- ... -->
</aws>
```

dynamoDB

`<dynamoDB>`Elemen mewakili kumpulan pengaturan untuk Amazon DynamoDB. Elemen ini dapat menyertakan atribut ConversionSchema, yang mewakili versi yang akan digunakan untuk mengkonversi antara objek .NET dan DynamoDB. Nilai yang diizinkan termasuk V1 dan V2. Atribut ini memetakan ke `Amazon.DynamoDBv2.DynamoDBEntryConversion` kelas di AWS SDK untuk .NET. Untuk informasi selengkapnya, lihat Seri DynamoDB - Skema Konversi.

Induk <dynamoDB> elemen adalah <aws> elemen.

<dynamoDB>Elemen dapat mencakup elemen <dynamoDBContext> anak.

Berikut ini adalah contoh <dynamoDB> elemen yang digunakan:

```
<dynamoDB
  conversionSchema="V2">
  <!-- ... -->
</dynamoDB>
```

dinamo DBContext

<dynamoDBContext>Elemen mewakili kumpulan pengaturan khusus konteks Amazon DynamoDB. Elemen ini dapat menyertakan tableNamePrefixatribut, yang mewakili awalan nama tabel default konteks DynamoDB akan digunakan jika tidak dikonfigurasi secara manual. Atribut ini memetakan ke `Amazon.Util.DynamoDBContextConfig.TableNamePrefix` `Amazon.AWSConfigs.DynamoDBConfig.Context.TableNamePrefix` properti dari properti di AWS SDK untuk .NET. Untuk informasi selengkapnya, lihat Penyempurnaan pada DynamoDB SDK.

Induk <dynamoDBContext> elemen adalah <dynamoDB> elemen.

<dynamoDBContext>Elemen dapat mencakup elemen anak berikut:

- <alias>(satu atau lebih contoh)
- <map>(satu atau lebih contoh)

Berikut ini adalah contoh <dynamoDBContext> elemen yang digunakan:

```
<dynamoDBContext
  tableNamePrefix="Test-">
  <!-- ... -->
</dynamoDBContext>
```

ec2

<ec2>Elemen mewakili kumpulan EC2 pengaturan Amazon. Elemen ini dapat mencakup atribut useSignatureVersion4, yang menentukan apakah tanda tangan versi 4 penandatanganan akan digunakan untuk semua permintaan (true) atau apakah tanda tangan versi 4

penandatanganan tidak akan digunakan untuk semua permintaan (false, default).
Atribut ini memetakan ke `Amazon.Util.EC2Config.UseSignatureVersion4` `Amazon.AWSConfigs.EC2Config.UseSignatureVersion4` properti dari properti di AWS SDK untuk .NET.

Induk `<ec2>` elemen adalah elemen.

`<ec2>`Elemen tidak mengandung elemen anak.

Berikut ini adalah contoh `<ec2>` elemen yang digunakan:

```
<ec2
  useSignatureVersion4="true" />
```

pencatatan log

`<logging>`Elemen mewakili kumpulan pengaturan untuk pencatatan respons dan pencatatan metrik kinerja. Elemen ini dapat mencakup atribut berikut:

`logMetrics`

Apakah metrik kinerja akan dicatat untuk semua klien dan konfigurasi (true); jika tidak, false. Atribut ini memetakan ke `Amazon.Util.LoggingConfig.LogMetrics` `Amazon.AWSConfigs.LoggingConfig.LogMetrics` properti dari properti di AWS SDK untuk .NET.

`logMetricsCustomFormatter`

Jenis data dan nama rakitan formatter kustom untuk metrik logging. Atribut ini memetakan ke `Amazon.Util.LoggingConfig.LogMetricsCustomFormatter` `Amazon.AWSConfigs.LoggingConfig.LogMetricsCustomFormatter` properti dari properti di AWS SDK untuk .NET.

`logMetricsFormat`

Format di mana metrik logging disajikan (peta ke `Amazon.Util.LoggingConfig.LogMetricsFormat` properti dari `Amazon.AWSConfigs.LoggingConfig.LogMetricsFormat` properti di AWS SDK untuk .NET).

Nilai yang diizinkan meliputi:

`JSON`

Gunakan format JSON.

`Standard`

Gunakan format default.

`logResponses`

Kapan harus mencatat tanggapan layanan (peta ke `Amazon.Util.LoggingConfig.LogResponses` properti dari `Amazon.AWSConfigs.LoggingConfig.LogResponses` properti di AWS SDK untuk .NET).

Nilai yang diizinkan meliputi:

`Always`

Selalu log tanggapan layanan.

`Never`

Jangan pernah mencatat tanggapan layanan.

`OnError`

Hanya log tanggapan layanan ketika ada kesalahan.

`logTo`

Tempat masuk (peta ke `LogTo` properti dari `Amazon.AWSConfigs.LoggingConfig.LogTo` properti di AWS SDK untuk .NET).

Nilai yang diizinkan mencakup satu atau lebih dari:

`Log4Net`

Masuk ke log4net.

`None`

Nonaktifkan logging.

`SystemDiagnostics`

Masuk ke System.Diagnostics.

Induk <logging> elemen adalah <aws> elemen.

<logging>Elemen tidak mengandung elemen anak.

Berikut ini adalah contoh <logging> elemen yang digunakan:

```
<logging
  logTo="SystemDiagnostics"
  logResponses="OnError"
  logMetrics="true"
  logMetricsFormat="JSON"
  logMetricsCustomFormatter="MyLib.Util.MyMetricsFormatter, MyLib" />
```

map

<map>Elemen mewakili satu item dalam koleksi type-to-table pemetaan dari tipe.NET ke tabel DynamoDB (peta ke instance `TypeMapping` kelas dari properti di). `Amazon.AWSConfigs.DynamoDBConfig.Context.TypeMappings` AWS SDK untuk .NET Untuk informasi selengkapnya, lihat Penyempurnaan pada DynamoDB SDK.

Elemen ini dapat mencakup atribut berikut:

`targetTable`

Tabel DynamoDB tempat pemetaan berlaku. Atribut ini memetakan ke `Amazon.Util.TypeMapping.TargetTable` properti di AWS SDK untuk .NET.

`type`

Jenis dan nama perakitan yang diterapkan pemetaan. Atribut ini memetakan ke `Amazon.Util.TypeMapping.Type` properti di AWS SDK untuk .NET.

Induk <map> elemen adalah <dynamoDBContext> elemen.

<map>Elemen dapat mencakup satu atau lebih contoh elemen <property> anak.

Berikut ini adalah contoh <map> elemen yang digunakan:

```
<map
  type="SampleApp.Models.Movie, SampleDLL"
  targetTable="Movies">
```

```
  <!-- ... -->
</map>
```

properti

<property>Elemen merupakan properti DynamoDB. (Elemen ini memetakan ke instance Amazon.Util. PropertyConfig class dari AddProperty metode dalam AWS SDK untuk .NET) Untuk informasi selengkapnya, lihat Penyempurnaan ke DynamoDB SDK dan DynamoDB Attribut.

Elemen ini dapat mencakup atribut berikut:

attribute

Nama atribut untuk properti, seperti nama kunci rentang. Atribut ini memetakan ke Amazon.Util.PropertyConfig.Attribute properti di AWS SDK untuk .NET.

converter

Jenis konverter yang harus digunakan untuk properti ini. Atribut ini memetakan ke Amazon.Util.PropertyConfig.Converter properti di AWS SDK untuk .NET.

ignore

Apakah properti terkait harus diabaikan (true); jika tidak, false. Atribut ini memetakan ke Amazon.Util.PropertyConfig.Ignore properti di AWS SDK untuk .NET.

name

Nama properti. Atribut ini memetakan ke Amazon.Util.PropertyConfig.Name properti di AWS SDK untuk .NET.

version

Apakah properti ini harus menyimpan nomor versi item (true); jika tidak, false. Atribut ini memetakan ke Amazon.Util.PropertyConfig.Version properti di AWS SDK untuk .NET.

Induk <property> elemen adalah <map> elemen.

<property>Elemen tidak mengandung elemen anak.

Berikut ini adalah contoh <property> elemen yang digunakan:

```
<property
```

```
  name="Rating"
  converter="SampleApp.Models.RatingConverter, SampleDLL" />
```

proxy

<proxy>Elemen mewakili pengaturan untuk mengkonfigurasi proxy AWS SDK untuk .NET untuk digunakan. Elemen ini dapat mencakup atribut berikut:

host

Nama host atau alamat IP dari server proxy. Atribut ini memetakan ke Amazon.Util.ProxyConfig.Host Amazon.AWSConfigs.ProxyConfig.Host properti dari properti di AWS SDK untuk .NET.

password

Kata sandi untuk mengautentikasi dengan server proxy. Atribut ini memetakan ke Amazon.Util.ProxyConfig.Password Amazon.AWSConfigs.ProxyConfig.Password properti dari properti di AWS SDK untuk .NET.

port

Nomor port proxy. Atribut ini memetakan ke Amazon.Util.ProxyConfig.Port Amazon.AWSConfigs.ProxyConfig.Port properti dari properti di AWS SDK untuk .NET.

nama pengguna

Nama pengguna untuk mengautentikasi dengan server proxy. Atribut ini memetakan ke Amazon.Util.ProxyConfig.Username Amazon.AWSConfigs.ProxyConfig.Username properti dari properti di AWS SDK untuk .NET.

Induk <proxy> elemen adalah <aws> elemen.

<proxy>Elemen tidak mengandung elemen anak.

Berikut ini adalah contoh <proxy> elemen yang digunakan:

```
<proxy
  host="192.0.2.0"
  port="1234"
  username="My-Username-Here"
```

```
  password="My-Password-Here" />
```

s3

<s3>Elemen mewakili kumpulan pengaturan Amazon S3. Elemen ini dapat mencakup atribut useSignatureVersion4, yang menentukan apakah tanda tangan versi 4 penandatanganan akan digunakan untuk semua permintaan (true) atau apakah tanda tangan versi 4 penandatanganan tidak akan digunakan untuk semua permintaan (false, default). Atribut ini memetakan ke `Amazon.AWSConfigs.S3Config.UseSignatureVersion4` properti di AWS SDK untuk .NET.

Induk <s3> elemen adalah <aws> elemen.

<s3>Elemen tidak mengandung elemen anak.

Berikut ini adalah contoh <s3> elemen yang digunakan:

```
<s3 useSignatureVersion4="true" />
```

Menggunakan kredensial warisan

Topik di bagian ini memberikan informasi tentang penggunaan kredensi jangka panjang atau jangka pendek tanpa menggunakan. AWS IAM Identity Center

> **Warning**
>
> Untuk menghindari risiko keamanan, jangan gunakan pengguna IAM 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.

> **Note**
>
> Informasi dalam topik ini adalah untuk keadaan di mana Anda perlu memperoleh dan mengelola kredensi jangka pendek atau jangka panjang secara manual. Untuk informasi tambahan tentang kredensi jangka pendek dan jangka panjang, lihat Cara lain untuk mengautentikasi di Panduan Referensi Alat AWS SDKs dan Alat.
> Untuk praktik keamanan terbaik, gunakan AWS IAM Identity Center, seperti yang dijelaskan dalamAutentikasi dengan AWS.

Peringatan penting dan panduan untuk kredensional

Peringatan untuk kredensional

- JANGAN gunakan kredensi root akun Anda untuk mengakses AWS sumber daya. Kredensi ini menyediakan akses akun yang tidak terbatas dan sulit dicabut.
- JANGAN menaruh kunci akses literal atau informasi kredensi dalam file aplikasi Anda. Jika Anda melakukannya, Anda membuat risiko secara tidak sengaja mengekspos kredensialnya jika, misalnya, Anda mengunggah proyek ke repositori publik.
- JANGAN sertakan file yang berisi kredensi di area proyek Anda.
- Ketahuilah bahwa setiap kredensional yang disimpan dalam AWS `credentials` file bersama, disimpan dalam teks biasa.

Panduan tambahan untuk mengelola kredensional dengan aman

Untuk diskusi umum tentang cara mengelola AWS kredensional dengan aman, lihat kredenal AWS keamanan dalam praktik Referensi Umum AWSdan kasus penggunaan terbaik Keamanan di Panduan Pengguna IAM. Selain diskusi tersebut, pertimbangkan hal berikut:

- Buat pengguna tambahan, seperti pengguna di IAM Identity Center, dan gunakan kredensialnya alih-alih menggunakan kredensi pengguna AWS root Anda. Kredensi untuk pengguna lain dapat dicabut jika perlu atau bersifat sementara. Selain itu, Anda dapat menerapkan kebijakan kepada setiap pengguna untuk akses hanya ke sumber daya dan tindakan tertentu dan dengan demikian mengambil sikap izin hak istimewa paling sedikit.

- Gunakan peran IAM untuk tugas untuk tugas Amazon Elastic Container Service (Amazon ECS).

- Gunakan peran IAM untuk aplikasi yang berjalan di instans Amazon EC2.

- Gunakan kredensi sementara atau variabel lingkungan untuk aplikasi yang tersedia bagi pengguna di luar organisasi Anda.

Topik

- Menggunakan file AWS kredensial bersama
- Menggunakan SDK Store (khusus Windows)

Menggunakan file AWS kredensial bersama

(Pastikan untuk meninjau peringatan dan panduan penting untuk kredensialnya.)

Salah satu cara untuk memberikan kredensi untuk aplikasi Anda adalah dengan membuat profil di file AWS kredensial bersama dan kemudian menyimpan kredensi di profil tersebut. File ini dapat digunakan oleh yang lain AWS SDKs. Hal ini juga dapat digunakan oleh AWS CLI, AWS Tools for Windows PowerShell, dan AWS toolkit untuk Visual Studio, JetBrains, dan VS Code.

> **Warning**
>
> Untuk menghindari risiko keamanan, jangan gunakan pengguna IAM 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.

> **Note**
>
> Informasi dalam topik ini adalah untuk keadaan di mana Anda perlu memperoleh dan mengelola kredensi jangka pendek atau jangka panjang secara manual. Untuk informasi tambahan tentang kredensi jangka pendek dan jangka panjang, lihat Cara lain untuk mengautentikasi di Panduan Referensi Alat AWS SDKs dan Alat.
> Untuk praktik keamanan terbaik, gunakan AWS IAM Identity Center, seperti yang dijelaskan dalamAutentikasi dengan AWS.

Informasi umum

Secara default, file AWS kredensi bersama terletak di direktori di dalam `.aws` direktori home Anda dan diberi nama`credentials`; yaitu, `~/.aws/credentials` (Linux atau macOS) atau `%USERPROFILE%\.aws\credentials` (Windows). Untuk informasi tentang lokasi alternatif, lihat Lokasi file bersama di Panduan Referensi Alat AWS SDKs dan. Lihat juga Mengakses kredensi dan profil dalam aplikasi.

File AWS kredensial bersama adalah file plaintext dan mengikuti format tertentu. Untuk informasi tentang format file AWS kredensional, lihat Format file kredensial di Panduan Referensi Alat AWS SDKs dan.

Anda dapat mengelola profil dalam file AWS kredensi bersama dengan beberapa cara.

- Gunakan editor teks apa pun untuk membuat dan memperbarui file AWS kredensi bersama.
- Gunakan Amazon.Runtime. CredentialManagementnamespace AWS SDK untuk .NET API, seperti yang ditunjukkan nanti dalam topik ini.
- Gunakan perintah dan prosedur untuk Alat AWS untuk PowerShelldan AWS toolkit untuk Visual Studio, JetBrains, dan VS Code.
- Gunakan AWS CLIperintah; misalnya, `aws configure set aws_access_key_id` dan`aws configure set aws_secret_access_key`.

Contoh manajemen profil

Bagian berikut menunjukkan contoh profil dalam file AWS kredensial bersama. Beberapa contoh menunjukkan hasilnya, yang dapat diperoleh melalui salah satu metode manajemen kredensional yang dijelaskan sebelumnya. Contoh lain menunjukkan cara menggunakan metode tertentu.

Profil default

File AWS kredensial bersama hampir selalu memiliki profil bernama default. Di sinilah AWS SDK untuk .NET mencari kredensi jika tidak ada profil lain yang ditentukan.

`[default]`Profil biasanya terlihat seperti berikut ini.

```
[default]
aws_access_key_id = AKIAIOSFODNN7EXAMPLE
aws_secret_access_key = wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

Buat profil secara terprogram

Contoh ini menunjukkan cara membuat profil dan menyimpannya ke file AWS kredensial bersama secara terprogram. Ini menggunakan kelas berikut dari Amazon.Runtime. CredentialManagementnamespace: CredentialProfileOptions, CredentialProfile, dan. SharedCredentialsFile

```
using Amazon.Runtime.CredentialManagement;
...

// Do not include credentials in your code.
WriteProfile("my_new_profile", SecurelyStoredKeyID, SecurelyStoredSecretAccessKey);
...
```

```
void WriteProfile(string profileName, string keyId, string secret)
{
    Console.WriteLine($"Create the [{profileName}] profile...");
    var options = new CredentialProfileOptions
    {
        AccessKey = keyId,
        SecretKey = secret
    };
    var profile = new CredentialProfile(profileName, options);
    var sharedFile = new SharedCredentialsFile();
    sharedFile.RegisterProfile(profile);
}
```

> **Warning**
>
> Kode seperti ini umumnya tidak boleh ada dalam aplikasi Anda. Jika Anda memasukkannya ke dalam aplikasi Anda, lakukan tindakan pencegahan yang tepat untuk memastikan bahwa kunci teks biasa tidak mungkin terlihat dalam kode, melalui jaringan, atau bahkan di memori komputer.

Berikut ini adalah profil yang dibuat oleh contoh ini.

```
[my_new_profile]
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

Perbarui profil yang ada secara terprogram

Contoh ini menunjukkan kepada Anda cara memperbarui profil yang dibuat sebelumnya secara terprogram. Ini menggunakan kelas berikut dari Amazon.Runtime.CredentialManagementnamespace: CredentialProfiledan. SharedCredentialsFile Ini juga menggunakan RegionEndpointkelas namespace Amazon.

```
using Amazon.Runtime.CredentialManagement;
...

AddRegion("my_new_profile", RegionEndpoint.USWest2);
...
```

```
void AddRegion(string profileName, RegionEndpoint region)
{
    var sharedFile = new SharedCredentialsFile();
    CredentialProfile profile;
    if (sharedFile.TryGetProfile(profileName, out profile))
    {
        profile.Region = region;
        sharedFile.RegisterProfile(profile);
    }
}
```

Berikut ini adalah profil yang diperbarui.

```
[my_new_profile]
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
region=us-west-2
```

> **Note**
>
> Anda juga dapat mengatur AWS Wilayah di lokasi lain dan dengan menggunakan metode lain. Lihat informasi yang lebih lengkap di Mengatur AWS Wilayah untuk AWS SDK untuk .NET.

Menggunakan SDK Store (khusus Windows)

(Pastikan untuk meninjau peringatan dan pedoman penting.)

Di Windows, SDK Store adalah tempat lain untuk membuat profil dan menyimpan kredensi terenkripsi untuk aplikasi Anda. AWS SDK untuk .NET Itu terletak di`%USERPROFILE%\AppData\Local\AWSToolkit\RegisteredAccounts.json`. Anda dapat menggunakan SDK Store selama pengembangan sebagai alternatif untuk file AWS kredensial bersama.

> **Warning**
>
> Untuk menghindari risiko keamanan, jangan gunakan pengguna IAM 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.

> **Note**
>
> Informasi dalam topik ini adalah untuk keadaan di mana Anda perlu memperoleh dan mengelola kredensi jangka pendek atau jangka panjang secara manual. Untuk informasi tambahan tentang kredensil jangka pendek dan jangka panjang, lihat Cara lain untuk mengautentikasi di Panduan Referensi Alat AWS SDKs dan Alat.
> Untuk praktik keamanan terbaik, gunakan AWS IAM Identity Center, seperti yang dijelaskan dalamAutentikasi dengan AWS.

Informasi umum

SDK Store memberikan manfaat berikut:

- Kredensil di SDK Store dienkripsi, dan SDK Store berada di direktori home pengguna. Ini membatasi risiko secara tidak sengaja mengekspos kredensil Anda.
- SDK Store juga menyediakan kredensil untuk dan. AWS Tools for Windows PowerShellAWS Toolkit for Visual Studio

Profil SDK Store khusus untuk pengguna tertentu pada host tertentu. Anda tidak dapat menyalinnya ke host lain atau pengguna lain. Ini berarti Anda tidak dapat menggunakan kembali profil SDK Store yang ada di mesin pengembangan Anda untuk host atau mesin pengembang lain. Ini juga berarti bahwa Anda tidak dapat menggunakan profil SDK Store dalam aplikasi produksi.

Anda dapat mengelola profil di SDK Store dengan cara berikut:

- Gunakan antarmuka pengguna grafis (GUI) di. AWS Toolkit for Visual Studio
- Gunakan Amazon.Runtime. CredentialManagementnamespace AWS SDK untuk .NET API, seperti yang ditunjukkan nanti dalam topik ini.
- Gunakan perintah dari AWS Tools for Windows PowerShell; misalnya, `Set-AWSCredential` dan`Remove-AWSCredentialProfile`.

Contoh manajemen profil

Contoh berikut menunjukkan cara membuat dan memperbarui profil secara terprogram di SDK Store.

Buat profil secara terprogram

Contoh ini menunjukkan cara membuat profil dan menyimpannya ke SDK Store secara terprogram. Ini menggunakan kelas berikut dari Amazon.Runtime. CredentialManagementnamespace: CredentialProfileOptions, CredentialProfile, dan Net SDKCredentials File.

```
using Amazon.Runtime.CredentialManagement;
...

// Do not include credentials in your code.
WriteProfile("my_new_profile", SecurelyStoredKeyID, SecurelyStoredSecretAccessKey);
...

void WriteProfile(string profileName, string keyId, string secret)
{
    Console.WriteLine($"Create the [{profileName}] profile...");
    var options = new CredentialProfileOptions
    {
        AccessKey = keyId,
        SecretKey = secret
    };
    var profile = new CredentialProfile(profileName, options);
    var netSdkStore = new NetSDKCredentialsFile();
    netSdkStore.RegisterProfile(profile);
}
```

> **Warning**
>
> Kode seperti ini umumnya tidak boleh ada dalam aplikasi Anda. Jika disertakan dalam aplikasi Anda, lakukan tindakan pencegahan yang tepat untuk memastikan bahwa kunci teks biasa tidak mungkin terlihat dalam kode, melalui jaringan, atau bahkan di memori komputer.

Berikut ini adalah profil yang dibuat oleh contoh ini.

```
"[generated GUID]" : {
    "AWSAccessKey" : "01000000D08...[etc., encrypted access key ID]",
    "AWSSecretKey" : "01000000D08...[etc., encrypted secret access key]",
    "ProfileType"  : "AWS",
    "DisplayName"  : "my_new_profile",
}
```

Perbarui profil yang ada secara terprogram

Contoh ini menunjukkan kepada Anda cara memperbarui profil yang dibuat sebelumnya secara terprogram. Ini menggunakan kelas berikut dari Amazon.Runtime.CredentialManagementnamespace: CredentialProfiledan Net SDKCredentials File. Ini juga menggunakan RegionEndpointkelas namespace Amazon.

```
using Amazon.Runtime.CredentialManagement;
...

AddRegion("my_new_profile", RegionEndpoint.USWest2);
...

void AddRegion(string profileName, RegionEndpoint region)
{
    var netSdkStore = new NetSDKCredentialsFile();
    CredentialProfile profile;
    if (netSdkStore.TryGetProfile(profileName, out profile))
    {
        profile.Region = region;
        netSdkStore.RegisterProfile(profile);
    }
}
```

Berikut ini adalah profil yang diperbarui.

```
"[generated GUID]" : {
    "AWSAccessKey" : "01000000D08...[etc., encrypted access key ID]",
    "AWSSecretKey" : "01000000D08...[etc., encrypted secret access key]",
    "ProfileType"  : "AWS",
    "DisplayName"  : "my_new_profile",
    "Region"       : "us-west-2"
}
```

> **Note**
>
> Anda juga dapat mengatur AWS Wilayah di lokasi lain dan dengan menggunakan metode lain. Lihat informasi yang lebih lengkap di Mengatur AWS Wilayah untuk AWS SDK untuk .NET.

Menggunakan AWS SDK untuk .NET

Bagian ini memberikan informasi tentang fitur AWS SDK untuk .NET yang mungkin perlu Anda pertimbangkan saat membuat aplikasi Anda.

Sebelum Anda mulai, pastikan Anda telah mengatur lingkungan Anda dan mengkonfigurasi proyek Anda.

Untuk informasi tentang mengembangkan perangkat lunak untuk AWS layanan tertentu bersama dengan contoh kode, lihat AWS Layanan panggilan. Untuk contoh kode tambahan, lihatSDK untuk .NET (v4) contoh kode.

Topik

- Pemrograman secara asinkron menggunakan AWS SDK untuk .NET
- Menggunakan hasil paginasi di AWS SDK untuk .NET
- Support untuk HTTP 2 di AWS SDK untuk .NET
- Alat tambahan

Pemrograman secara asinkron menggunakan AWS SDK untuk .NET

AWS SDK untuk .NET Menggunakan Task-based Asynchronous Pattern (TAP) untuk implementasi asinkron. Untuk mempelajari TAP selengkapnya, lihat Pola Asinkron Berbasis Tugas (TAP) di docs.microsoft.com.

Topik ini memberi Anda gambaran umum tentang cara menggunakan TAP dalam panggilan Anda ke klien AWS layanan.

Metode asinkron dalam AWS SDK untuk .NET API adalah operasi berdasarkan `Task` kelas atau kelas. `Task<TResult>` Lihat docs.microsoft.com untuk informasi tentang kelas-kelas ini: Kelas tugas, Tugas < > kelas. TResult

Ketika metode API ini dipanggil dalam kode Anda, mereka harus dipanggil dalam fungsi yang dideklarasikan dengan `async` kata kunci, seperti yang ditunjukkan pada contoh berikut.

```
static async Task Main(string[] args)
```

```
{
  ...
  // Call the function that contains the asynchronous API method.
  // Could also call the asynchronous API method directly from Main
  //  because Main is declared async
  var response = await ListBucketsAsync();
  Console.WriteLine($"Number of buckets: {response.Buckets.Count}");
  ...
}

// Async method to get a list of Amazon S3 buckets.
private static async Task<ListBucketsResponse> ListBucketsAsync()
{
  ...
  var response = await s3Client.ListBucketsAsync();
  return response;
}
```

Seperti yang ditunjukkan pada cuplikan kode sebelumnya, ruang lingkup yang lebih disukai untuk `async` deklarasi adalah fungsinya. `Main` Menyetel `async` cakupan ini memastikan bahwa semua panggilan ke klien AWS layanan harus asinkron. Jika Anda tidak dapat mendeklarasikan `Main` asinkron karena alasan tertentu, Anda dapat menggunakan `async` kata kunci pada fungsi selain `Main` dan kemudian memanggil metode API dari sana, seperti yang ditunjukkan pada contoh berikut.

```
static void Main(string[] args)
{
  ...
  Task<ListBucketsResponse> response = ListBucketsAsync();
  Console.WriteLine($"Number of buckets: {response.Result.Buckets.Count}");
  ...
}

// Async method to get a list of Amazon S3 buckets.
private static async Task<ListBucketsResponse> ListBucketsAsync()
{
  ...
  var response = await s3Client.ListBucketsAsync();
  return response;
}
```

Perhatikan `Task<>` sintaks khusus yang diperlukan `Main` saat Anda menggunakan pola ini. Selain itu, Anda harus menggunakan **`Result`**anggota respons untuk mendapatkan data.

Anda dapat melihat contoh lengkap panggilan asinkron ke klien AWS layanan di Membuat aplikasi sederhana bagian (Aplikasi lintas platform sederhanadanAplikasi berbasis Windows sederhana) dan di. Contoh kode terpandu

Menggunakan hasil paginasi di AWS SDK untuk .NET

Beberapa AWS layanan mengumpulkan dan menyimpan sejumlah besar data, yang dapat Anda ambil dengan menggunakan panggilan API. AWS SDK untuk .NETJika jumlah data yang ingin Anda ambil menjadi terlalu besar untuk satu panggilan API, Anda dapat memecah hasilnya menjadi potongan-potongan yang lebih mudah dikelola melalui penggunaan pagination.

Untuk memungkinkan Anda melakukan pagination, objek permintaan dan respons untuk banyak klien layanan di SDK menyediakan token lanjutan (biasanya bernama). `NextToken` Beberapa klien layanan ini juga menyediakan paginator.

Paginator memungkinkan Anda menghindari overhead token lanjutan, yang mungkin melibatkan loop, variabel status, beberapa panggilan API, dan sebagainya. Bila Anda menggunakan paginator, Anda dapat mengambil data dari AWS layanan melalui satu baris kode, deklarasi `foreach` loop. Jika beberapa panggilan API diperlukan untuk mengambil data, paginator menangani ini untuk Anda.

Di mana saya menemukan paginator?

Tidak semua layanan menyediakan paginator. Salah satu cara untuk menentukan apakah layanan menyediakan paginator untuk API tertentu adalah dengan melihat definisi kelas klien layanan di Referensi AWS SDK untuk .NET API.

Misalnya, jika Anda memeriksa definisi untuk AmazonCloudWatchLogsClientkelas, Anda melihat `Paginators` properti. Ini adalah properti yang menyediakan paginator untuk Amazon CloudWatch Logs.

Apa yang diberikan paginator kepada saya?

Paginator berisi properti yang memungkinkan Anda melihat respons lengkap. Mereka juga biasanya berisi satu atau lebih properti yang memungkinkan Anda mengakses bagian paling menarik dari tanggapan, yang akan kami sebut hasil utama.

Misalnya, dalam yang `AmazonCloudWatchLogsClient` disebutkan sebelumnya, `Paginator` objek berisi `Responses` properti dengan DescribeLogGroupsResponseobjek lengkap dari panggilan API. `Responses`Properti ini berisi, antara lain, kumpulan grup log.

Objek Paginator juga berisi satu hasil kunci bernama. `LogGroups` Properti ini hanya menyimpan bagian log grup dari respons. Memiliki hasil kunci ini memungkinkan Anda untuk mengurangi dan menyederhanakan kode Anda dalam banyak keadaan.

Pagination sinkron vs asinkron

Paginator menyediakan mekanisme sinkron dan asinkron untuk pagination. Pagination sinkron tersedia dalam proyek .NET Framework 4.7.2 (atau yang lebih baru). Pagination asinkron tersedia di proyek .NET Core (.NET Core 3.1, .NET 5, dan seterusnya).

Karena operasi asinkron dan .NET Core direkomendasikan, contoh yang muncul berikutnya menunjukkan pagination asinkron. Informasi tentang cara melakukan tugas yang sama menggunakan pagination sinkron dan .NET Framework 4.7.2 (atau yang lebih baru) ditampilkan setelah contoh di. Pertimbangan tambahan untuk paginator

Contoh

Contoh berikut menunjukkan cara menggunakan AWS SDK untuk .NET untuk menampilkan daftar grup log. Sebagai kontras, contoh menunjukkan bagaimana melakukan ini baik dengan maupun tanpa paginator. Sebelum melihat kode lengkap, ditampilkan nanti, pertimbangkan cuplikan berikut.

Mendapatkan grup CloudWatch log tanpa paginator

```
      // Loop as many times as needed to get all the log groups
      var request = new DescribeLogGroupsRequest{Limit = LogGroupLimit};
      do
      {
        Console.WriteLine($"Getting up to {LogGroupLimit} log groups...");
        var response = await cwClient.DescribeLogGroupsAsync(request);
        foreach(var logGroup in response.LogGroups)
        {
          Console.WriteLine($"{logGroup.LogGroupName}");
        }
        request.NextToken = response.NextToken;
      } while(!string.IsNullOrEmpty(request.NextToken));
```

Mendapatkan grup CloudWatch log dengan menggunakan paginator

```
      // No need to loop to get all the log groups--the SDK does it for us behind the
 scenes
```

```
    var paginatorForLogGroups =
      cwClient.Paginators.DescribeLogGroups(new DescribeLogGroupsRequest());
    await foreach(var logGroup in paginatorForLogGroups.LogGroups)
    {
      Console.WriteLine(logGroup.LogGroupName);
    }
```

Hasil dari kedua cuplikan ini persis sama, sehingga keuntungan dalam menggunakan paginator dapat dilihat dengan jelas.

> **Note**
>
> Sebelum Anda mencoba membangun dan menjalankan kode lengkap, pastikan Anda telah menyiapkan lingkungan dan proyek Anda.
> Anda mungkin juga membutuhkan Microsoft.Bcl. AsyncInterfaces NuGet paket karena paginator asinkron menggunakan antarmuka. `IAsyncEnumerable`

Kode lengkap

Bagian ini menunjukkan referensi yang relevan dan kode lengkap untuk contoh ini.

Referensi SDK

NuGet paket:

- AWSSDK.CloudWatch

Elemen pemrograman:

- Namespace Amazon. CloudWatch

  Kelas AmazonCloudWatchLogsClient
- Namespace Amazon. CloudWatchLogs.Model

  Kelas DescribeLogGroupsRequest

  Kelas DescribeLogGroupsResponse

  Kelas LogGroup

Kode lengkap

```
using System;
using System.Threading.Tasks;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;

namespace CWGetLogGroups
{
  class Program
  {
    // A small limit for demonstration purposes
    private const int LogGroupLimit = 3;

    //
    // Main method
    static async Task Main(string[] args)
    {
      var cwClient = new AmazonCloudWatchLogsClient();
      await DisplayLogGroupsWithoutPaginators(cwClient);
      await DisplayLogGroupsWithPaginators(cwClient);
    }



    //
    // Method to get CloudWatch log groups without paginators
    private static async Task DisplayLogGroupsWithoutPaginators(IAmazonCloudWatchLogs
 cwClient)
    {
      Console.WriteLine("\nGetting list of CloudWatch log groups without using
 paginators...");

 Console.WriteLine("-----------------------------------------------------------------");

      // Loop as many times as needed to get all the log groups
      var request = new DescribeLogGroupsRequest{Limit = LogGroupLimit};
      do
      {
        Console.WriteLine($"Getting up to {LogGroupLimit} log groups...");
        DescribeLogGroupsResponse response = await
 cwClient.DescribeLogGroupsAsync(request);
        foreach(LogGroup logGroup in response.LogGroups)
        {
          Console.WriteLine($"{logGroup.LogGroupName}");
```

```
        }
        request.NextToken = response.NextToken;
      } while(!string.IsNullOrEmpty(request.NextToken));
    }


    //
    // Method to get CloudWatch log groups by using paginators
    private static async Task DisplayLogGroupsWithPaginators(IAmazonCloudWatchLogs
cwClient)
    {
      Console.WriteLine("\nGetting list of CloudWatch log groups by using
paginators...");

Console.WriteLine("-------------------------------------------------------------");

      // Access the key results; i.e., the log groups
      // No need to loop to get all the log groups--the SDK does it for us behind the
scenes
      Console.WriteLine("\nFrom the key results...");
      Console.WriteLine("------------------------");
      IDescribeLogGroupsPaginator paginatorForLogGroups =
        cwClient.Paginators.DescribeLogGroups(new DescribeLogGroupsRequest());
      await foreach(LogGroup logGroup in paginatorForLogGroups.LogGroups)
      {
        Console.WriteLine(logGroup.LogGroupName);
      }

      // Access the full response
      // Create a new paginator, do NOT reuse the one from above
      Console.WriteLine("\nFrom the full response...");
      Console.WriteLine("-------------------------");
      IDescribeLogGroupsPaginator paginatorForResponses =
        cwClient.Paginators.DescribeLogGroups(new DescribeLogGroupsRequest());
      await foreach(DescribeLogGroupsResponse response in
paginatorForResponses.Responses)
      {
        Console.WriteLine($"Content length: {response.ContentLength}");
        Console.WriteLine($"HTTP result: {response.HttpStatusCode}");
        Console.WriteLine($"Metadata: {response.ResponseMetadata}");
        Console.WriteLine("Log groups:");
        foreach(LogGroup logGroup in response.LogGroups)
        {
          Console.WriteLine($"\t{logGroup.LogGroupName}");
```

```
        }
      }
    }
  }
}
```

Pertimbangan tambahan untuk paginator

- Paginator tidak dapat digunakan lebih dari sekali

  Jika Anda membutuhkan hasil AWS paginator tertentu di beberapa lokasi dalam kode Anda, Anda tidak boleh menggunakan objek paginator lebih dari sekali. Sebagai gantinya, buat paginator baru setiap kali Anda membutuhkannya. Konsep ini ditunjukkan dalam contoh kode sebelumnya dalam metode. `DisplayLogGroupsWithPaginators`

- pagination sinkron

  Pagination sinkron tersedia untuk proyek-proyek .NET Framework 4.7.2 (atau yang lebih baru).

  Untuk melihat ini, buat proyek .NET Framework 4.7.2 (atau yang lebih baru) dan salin kode sebelumnya ke sana. Kemudian cukup hapus `await` kata kunci dari dua panggilan `foreach` paginator, seperti yang ditunjukkan pada contoh berikut.

  ```
  /*await*/ foreach(var logGroup in paginatorForLogGroups.LogGroups)
  {
    Console.WriteLine(logGroup.LogGroupName);
  }
  ```

  Bangun dan jalankan proyek untuk melihat hasil yang sama yang Anda lihat dengan pagination asinkron.

Support untuk HTTP 2 di AWS SDK untuk .NET

Beberapa AWS layanan dan operasi memerlukan HTTP 2. Misalnya, streaming dua arah di Amazon Transcribe Streaming tidak dimungkinkan melalui HTTP 1.1 sehingga memerlukan HTTP 2 sebagai gantinya. Versi 4 dari dukungan AWS SDK untuk .NET tambahan untuk HTTP 2 sehingga Anda dapat menggunakan operasi ini dalam aplikasi Anda. Untuk operasi HTTP 2 dua arah, perilaku

SDK saat menerima aliran mirip dengan HTTP 1.1. Artinya, ketika aplikasi yang menggunakan SDK mengirim peristiwa ke layanan, permintaan memiliki penerbit yang ditetapkan oleh pengembang.

Untuk melihat perilaku ini beraksi, pertimbangkan contoh berikut untuk Amazon Transcribe Streaming. Ini menggunakan Amazon. TranscribeStreamingdan Amazon. TranscribeStreamingRuang nama .Model.

Dalam contoh ini, pengembang mendefinisikan `StartStreamTranscriptionRequest.AudioStreamPublisher` properti dengan fungsi callback, yang merupakan .NET. `Func` SDK menggunakan `Func` definisi for `AudioStreamPublisher` untuk menarik peristiwa dari kode pengguna untuk streaming ke pengguna. SDK memanggil `Func` sampai mengembalikan null.

Kode menunjukkan bagaimana audio dari file dapat dialirkan ke Amazon Transcribe Streaming untuk diproses.

```
using Amazon;
using Amazon.TranscribeStreaming;
using Amazon.TranscribeStreaming.Model;

CancellationTokenSource cancelSource = new CancellationTokenSource();

var client = new AmazonTranscribeStreamingClient(RegionEndpoint.USEast1);

var startRequest = new StartStreamTranscriptionRequest
{
    LanguageCode = LanguageCode.EnUS,
    MediaEncoding = MediaEncoding.Flac,
    MediaSampleRateHertz = 44100,
    NumberOfChannels = 2,
    EnableChannelIdentification = true
};

Stream fileStream = File.OpenRead("hello-world.flac");
var buffer = new byte[1024 * 10];
startRequest.AudioStreamPublisher += async () =>
{
    var bytesRead = await fileStream.ReadAsync(buffer, 0, buffer.Length);

    if (bytesRead == 0)
        return null;
```

```
    var audioEvent = new AudioEvent
    {
        AudioChunk = new MemoryStream(buffer, 0, bytesRead)
    };

    return audioEvent;
};

using var response = await client.StartStreamTranscriptionAsync(startRequest);
Console.WriteLine(response.HttpStatusCode);

response.TranscriptResultStream.ExceptionReceived +=
 TranscriptResultStream_ExceptionReceived;
response.TranscriptResultStream.TranscriptEventReceived +=
 TranscriptResultStream_TranscriptEventReceived;

void TranscriptResultStream_ExceptionReceived(object? sender,
 Amazon.Runtime.EventStreams.EventStreamExceptionReceivedArgs<TranscribeStreamingEventStreamExc
 e)
{
    Console.WriteLine(e.EventStreamException.Message);
    cancelSource.Cancel();
}
void TranscriptResultStream_TranscriptEventReceived(object? sender,
 Amazon.Runtime.EventStreams.EventStreamEventReceivedArgs<TranscriptEvent> e)
{
    foreach (var result in e.EventStreamEvent.Transcript.Results)
    {
        if (!string.Equals("ch_0", result.ChannelId,
 StringComparison.OrdinalIgnoreCase))
            continue;

        var text = result.Alternatives[0].Transcript;
        if (!string.IsNullOrEmpty(text))
        {
            Console.WriteLine(text);
        }
    }
}

_ = response.TranscriptResultStream.StartProcessingAsync();

try
{
```

```
    await Task.Delay(10000, cancelSource.Token);
}
catch (TaskCanceledException) { }
```

> **Warning**
>
> Beberapa operasi HTTP 2 dua arah, seperti metode `InvokeModelWithBidirectionalStreamAsync` dari Amazon Bedrock, Amazon.BedrockRuntimenamespace, jangan mengembalikan respons dari pemanggilan awal pada klien layanan sampai beberapa peristiwa telah diterbitkan. Perilaku ini dapat menyebabkan aplikasi Anda diblokir. Untuk menghindari situasi ini, pisahkan kode aplikasi yang menyediakan peristiwa ke penerbit, dan jalankan pada utas yang berbeda dari utas yang memanggil operasi pada klien layanan.

Pertimbangan tambahan

- AWS SDK untuk .NET dukungan untuk HTTP 2 hanya tersedia dalam versi yang menargetkan .NET 8 dan di atasnya. Ini tidak tersedia dalam versi yang menargetkan .NET Framework.
- Untuk informasi lebih rinci, lihat PR 3730 di aws-sdk-net GitHubrepositori.

Alat tambahan

Berikut ini adalah beberapa alat tambahan yang dapat Anda gunakan untuk memudahkan pekerjaan mengembangkan, menyebarkan, dan memelihara aplikasi.NET Anda.

AWS Menyebarkan Alat

Setelah Anda mengembangkan aplikasi .NET Core cloud-native Anda pada mesin pengembangan, Anda dapat menggunakan Alat AWS Deploy untuk.NET CLI agar lebih mudah menyebarkan aplikasi Anda. AWS

Untuk informasi selengkapnya, lihat Menyebarkan aplikasi ke AWS.

AWS Kerangka Pemrosesan Pesan untuk.NET

Jika Anda menggunakan layanan seperti Amazon SQS, Amazon SNS atau EventBridge Amazon, Anda mungkin dapat memanfaatkan Kerangka Pemrosesan Pesan AWS untuk.NET. Untuk informasi selengkapnya, lihat AWS Kerangka Pemrosesan Pesan untuk.NET.

Integrasi dengan .NET Aspire

Anda dapat memanfaatkan integrasi dengan.NET Aspire untuk meningkatkan loop dev dalam. Lihat informasi yang lebih lengkap di Integrasi AWS dengan .NET Aspire di AWS SDK untuk .NET.

Otentikasi dan otorisasi lanjutan dengan AWS SDK untuk .NET

Topik di bagian ini memberikan informasi tentang teknik lanjutan untuk otentikasi dan otorisasi dalam aplikasi Anda AWS SDK untuk .NET .

Topik

- Single sign-on dengan AWS SDK untuk .NET

Single sign-on dengan AWS SDK untuk .NET

AWS IAM Identity Center adalah layanan single sign-on (SSO) berbasis cloud yang memudahkan pengelolaan akses SSO secara terpusat ke semua aplikasi Anda dan cloud. Akun AWS Untuk detail selengkapnya, lihat Panduan Pengguna Pusat Identitas IAM.

Jika Anda tidak terbiasa dengan cara SDK berinteraksi dengan IAM Identity Center, lihat informasi berikut.

Pola interaksi tingkat tinggi

Pada tingkat tinggi, SDKs berinteraksi dengan IAM Identity Center dengan cara yang mirip dengan pola berikut:

1. IAM Identity Center dikonfigurasi, biasanya melalui konsol IAM Identity Center, dan pengguna SSO diundang untuk berpartisipasi.
2. AWS `config`File bersama di komputer pengguna diperbarui dengan informasi SSO.
3. Pengguna masuk melalui IAM Identity Center dan diberikan kredensi jangka pendek untuk izin AWS Identity and Access Management (IAM) yang telah dikonfigurasi untuk mereka. Masuk ini dapat dimulai melalui alat non-SDK seperti AWS CLI, atau secara terprogram melalui aplikasi.NET.
4. Pengguna melanjutkan untuk melakukan pekerjaan mereka. Ketika mereka menjalankan aplikasi lain yang menggunakan SSO, mereka tidak perlu masuk lagi untuk membuka aplikasi.

Sisa topik ini memberikan informasi referensi untuk pengaturan dan penggunaan AWS IAM Identity Center. Ini memberikan informasi tambahan dan lebih maju daripada pengaturan SSO dasar di.

Autentikasi dengan AWS Jika Anda baru mengenal SSO AWS, Anda mungkin ingin melihat topik itu terlebih dahulu untuk informasi mendasar, dan kemudian pada tutorial berikut untuk melihat SSO beraksi:

- Tutorial: Aplikasi .NET saja
- Tutorial: AWS CLI dan aplikasi.NET

Topik ini berisi bagian-bagian berikut:

- Prasyarat
- Menyiapkan profil SSO
- Menghasilkan dan menggunakan token SSO
- Sumber daya tambahan
- Tutorial

Prasyarat

Sebelum menggunakan IAM Identity Center, Anda harus melakukan tugas-tugas tertentu, seperti memilih sumber identitas dan mengkonfigurasi yang relevan Akun AWS dan aplikasi. Untuk informasi tambahan, lihat hal berikut:

- Untuk informasi umum tentang tugas-tugas ini, lihat Memulai di Panduan Pengguna Pusat Identitas IAM.
- Untuk contoh tugas tertentu, lihat daftar tutorial di akhir topik ini. Namun, pastikan untuk meninjau informasi dalam topik ini sebelum mencoba tutorial.

Menyiapkan profil SSO

Setelah Pusat Identitas IAM dikonfigurasi dalam yang relevan Akun AWS, profil bernama untuk SSO harus ditambahkan ke file bersama AWS `config` pengguna. Profil ini digunakan untuk terhubung ke portal AWS akses, yang mengembalikan kredensi jangka pendek untuk izin IAM yang telah dikonfigurasi untuk pengguna.

`config`File bersama biasanya dinamai `%USERPROFILE%\.aws\config` di Windows dan `~/.aws/config` di Linux dan macOS. Anda dapat menggunakan editor teks pilihan Anda untuk

menambahkan profil baru untuk SSO. Atau, Anda dapat menggunakan `aws configure sso` perintah. Untuk informasi selengkapnya tentang perintah ini, lihat Mengkonfigurasi AWS CLI untuk menggunakan Pusat Identitas IAM di AWS Command Line Interface Panduan Pengguna.

Profil baru ini mirip dengan yang berikut:

```
[profile my-sso-profile]
sso_start_url = https://my-sso-portal.awsapps.com/start
sso_region = us-west-2
sso_account_id = 123456789012
sso_role_name = SSOReadOnlyRole
```

Pengaturan untuk profil baru didefinisikan di bawah ini. Dua pengaturan pertama menentukan portal AWS akses. Dua pengaturan lainnya adalah pasangan yang, secara bersama-sama, menentukan izin yang telah dikonfigurasi untuk pengguna. Keempat pengaturan diperlukan.

`sso_start_url`

URL yang mengarah ke portal AWS akses organisasi. Untuk menemukan nilai ini, buka konsol Pusat Identitas IAM, pilih Pengaturan, dan temukan URL portal.

`sso_region`

Wilayah AWS Yang berisi host portal akses. Ini adalah Wilayah yang dipilih saat Anda mengaktifkan Pusat Identitas IAM. Ini bisa berbeda dari Wilayah yang Anda gunakan untuk tugas lain.

Untuk daftar lengkap Wilayah AWS dan kodenya, lihat Titik Akhir Regional di. Referensi Umum Amazon Web

`sso_account_id`

ID dari sebuah Akun AWS yang ditambahkan melalui AWS Organizations layanan. Untuk melihat daftar akun yang tersedia, buka konsol Pusat Identitas IAM dan buka Akun AWShalaman. ID akun yang Anda pilih untuk pengaturan ini akan sesuai dengan nilai yang Anda rencanakan untuk diberikan ke `sso_role_name` pengaturan, yang ditampilkan berikutnya.

`sso_role_name`

Nama set izin Pusat Identitas IAM. Set izin ini mendefinisikan izin yang diberikan pengguna melalui IAM Identity Center.

Prosedur berikut adalah salah satu cara untuk menemukan nilai untuk pengaturan ini.

1. Buka konsol Pusat Identitas IAM dan buka Akun AWShalaman.
2. Pilih akun untuk menampilkan detailnya. Akun yang Anda pilih akan menjadi akun yang berisi pengguna atau grup SSO yang ingin Anda berikan izin SSO.
3. Lihatlah daftar pengguna dan grup yang ditugaskan ke akun dan temukan pengguna atau grup yang diminati. Set izin yang Anda tentukan dalam `sso_role_name` pengaturan adalah salah satu set yang terkait dengan pengguna atau grup ini.

Saat memberikan nilai pada setelan ini, gunakan nama set izin, bukan Nama Sumber Daya Amazon (ARN).

Set izin memiliki kebijakan IAM dan kebijakan izin khusus yang dilampirkan padanya. Untuk informasi selengkapnya, lihat Set izin di Panduan Pengguna Pusat Identitas IAM.

Menghasilkan dan menggunakan token SSO

Untuk menggunakan SSO, pengguna harus terlebih dahulu membuat token sementara dan kemudian menggunakan token itu untuk mengakses AWS aplikasi dan sumber daya yang sesuai. Untuk aplikasi.NET, Anda dapat menggunakan metode berikut untuk menghasilkan dan menggunakan token sementara ini:

- Buat aplikasi.NET yang menghasilkan token terlebih dahulu, jika perlu, lalu gunakan token.
- Hasilkan token dengan AWS CLI dan kemudian gunakan token di aplikasi.NET.

Metode ini dijelaskan di bagian berikut dan ditunjukkan dalam tutorial.

> **Important**
>
> Aplikasi Anda harus mereferensikan NuGet paket-paket berikut agar resolusi SSO dapat berfungsi:
>
> - `AWSSDK.SSO`
> - `AWSSDK.SSOOIDC`
>
> Kegagalan untuk mereferensikan paket-paket ini akan menghasilkan pengecualian runtime.

Hanya aplikasi.NET

Bagian ini menunjukkan cara membuat aplikasi.NET yang menghasilkan token SSO sementara, jika perlu, dan kemudian menggunakan token itu. Untuk tutorial lengkap tentang proses ini, lihatTutorial untuk SSO hanya menggunakan aplikasi.NET.

Hasilkan dan gunakan token SSO secara terprogram

Selain menggunakan AWS CLI, Anda juga dapat menghasilkan token SSO secara terprogram.

Untuk melakukan ini, aplikasi Anda membuat `AWSCredentials`objek untuk profil SSO, yang memuat kredensi sementara jika ada yang tersedia. Kemudian, aplikasi Anda harus mentransmisikan `AWSCredentials` objek ke `SSOAWSCredentials`objek dan menyetel beberapa properti Options, termasuk metode callback yang digunakan untuk meminta pengguna informasi login, jika perlu.

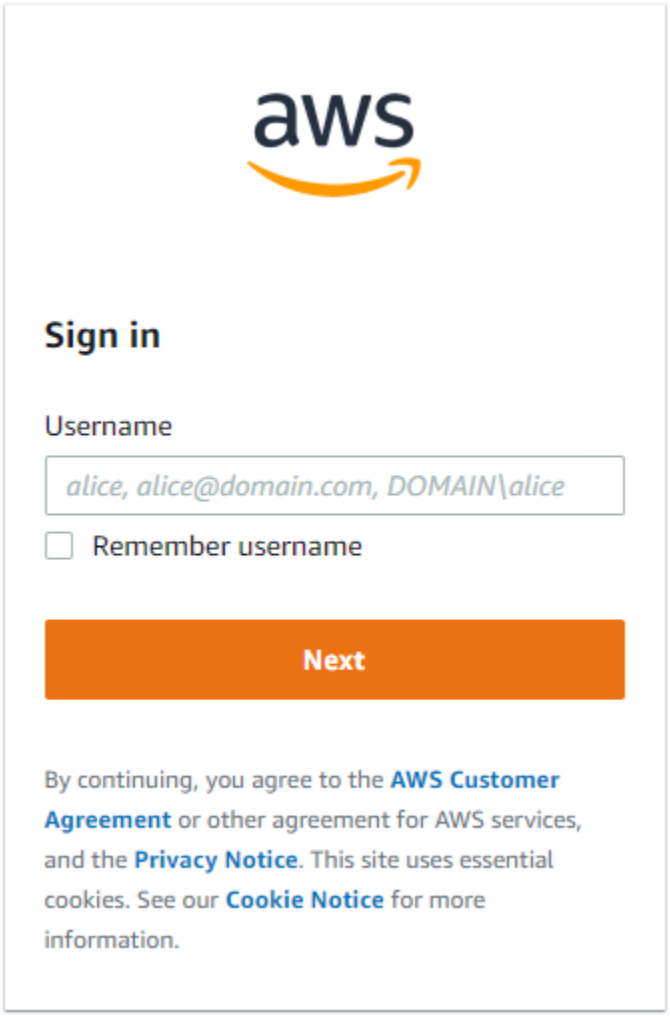
Metode ini ditunjukkan dalam cuplikan kode berikut.

> **Important**
>
> Aplikasi Anda harus mereferensikan NuGet paket-paket berikut agar resolusi SSO dapat berfungsi:
>
> - `AWSSDK.SSO`
> - `AWSSDK.SSOOIDC`
>
> Kegagalan untuk mereferensikan paket-paket ini akan menghasilkan pengecualian runtime.

```
static AWSCredentials LoadSsoCredentials()
{
    var chain = new CredentialProfileStoreChain();
    if (!chain.TryGetAWSCredentials("my-sso-profile", out var credentials))
        throw new Exception("Failed to find the my-sso-profile profile");

    var ssoCredentials = credentials as SSOAWSCredentials;

    ssoCredentials.Options.ClientName = "Example-SSO-App";
    ssoCredentials.Options.SsoVerificationCallback = args =>
    {
        // Launch a browser window that prompts the SSO user to complete an SSO sign-
in.
```

```
        // This method is only invoked if the session doesn't already have a valid SSO
 token.
        // NOTE: Process.Start might not support launching a browser on macOS or Linux.
 If not,
        //       use an appropriate mechanism on those systems instead.
        Process.Start(new ProcessStartInfo
        {
            FileName = args.VerificationUriComplete,
            UseShellExecute = true
        });
    };
    ssoCredentials.Options.SupportsGettingNewToken = true;

    return ssoCredentials;
}
```

Jika token SSO yang sesuai tidak tersedia, jendela browser default diluncurkan dan halaman login yang sesuai dibuka. Misalnya, jika Anda menggunakan Pusat Identitas IAM sebagai sumber Identitas, pengguna akan melihat halaman login yang mirip dengan berikut ini:



> **Note**
>
> String teks yang Anda sediakan tidak `SSOAWSCredentials.Options.ClientName` dapat memiliki spasi. Jika string memang memiliki spasi, Anda akan mendapatkan pengecualian runtime.

Tutorial untuk SSO hanya menggunakan aplikasi.NET

AWS CLI dan aplikasi.NET

Bagian ini menunjukkan kepada Anda cara membuat token SSO sementara dengan menggunakan AWS CLI, dan cara menggunakan token itu dalam aplikasi. Untuk tutorial lengkap tentang proses ini, lihatTutorial untuk SSO menggunakan aplikasi AWS CLI dan .NET.

Menghasilkan token SSO dengan menggunakan AWS CLI

Selain menghasilkan token SSO sementara secara terprogram, Anda menggunakan token AWS CLI untuk menghasilkan token. Informasi berikut menunjukkan caranya.

Setelah pengguna membuat profil berkemampuan SSO seperti yang ditunjukkan di bagian sebelumnya, mereka menjalankan `aws sso login` perintah dari file. AWS CLI Mereka harus yakin untuk menyertakan `--profile` parameter dengan nama profil berkemampuan SSO. Ini ditunjukkan dalam contoh berikut:

```
aws sso login --profile my-sso-profile
```

Jika pengguna ingin membuat token sementara baru setelah token saat ini kedaluwarsa, mereka dapat menjalankan perintah yang sama lagi.

Gunakan token SSO yang dihasilkan dalam aplikasi.NET

Informasi berikut menunjukkan kepada Anda cara menggunakan token sementara yang telah dibuat.

> **Important**
>
> Aplikasi Anda harus mereferensikan NuGet paket-paket berikut agar resolusi SSO dapat berfungsi:
>
> - `AWSSDK.SSO`
> - `AWSSDK.SSOOIDC`
>
> Kegagalan untuk mereferensikan paket-paket ini akan menghasilkan pengecualian runtime.

Aplikasi Anda membuat `AWSCredentials`objek untuk profil SSO, yang memuat kredenal sementara yang dihasilkan sebelumnya oleh file. AWS CLI Ini mirip dengan metode yang ditunjukkan dalam Mengakses kredensi dan profil dalam aplikasi dan memiliki bentuk berikut:

```
static AWSCredentials LoadSsoCredentials()
{
    var chain = new CredentialProfileStoreChain();
    if (!chain.TryGetAWSCredentials("my-sso-profile", out var credentials))
        throw new Exception("Failed to find the my-sso-profile profile");
```

```
    return credentials;
}
```

`AWSCredentials`Objek kemudian diteruskan ke konstruktor untuk klien layanan. Misalnya:

```
var S3Client_SSO = new AmazonS3Client(LoadSsoCredentials());
```

> **Note**
>
> Menggunakan `AWSCredentials` untuk memuat kredensil sementara tidak diperlukan jika aplikasi Anda telah dibuat untuk menggunakan `[default]` profil untuk SSO. Dalam hal ini, aplikasi dapat membuat klien AWS layanan tanpa parameter, mirip dengan "`var client = new AmazonS3Client();`".

Tutorial untuk SSO menggunakan aplikasi AWS CLI dan .NET

Sumber daya tambahan

Untuk bantuan tambahan, lihat sumber daya berikut.

- Apa itu Pusat Identitas IAM?
- Mengkonfigurasi AWS CLI untuk menggunakan IAM Identity Center
- Menggunakan kredensil Pusat Identitas IAM di AWS Toolkit for Visual Studio

Tutorial

Topik

- Tutorial untuk SSO hanya menggunakan aplikasi.NET
- Tutorial untuk SSO menggunakan aplikasi AWS CLI dan .NET

Tutorial untuk SSO hanya menggunakan aplikasi.NET

Tutorial ini menunjukkan cara mengaktifkan SSO untuk aplikasi dasar dan pengguna SSO uji. Ini mengkonfigurasi aplikasi untuk menghasilkan token SSO sementara secara terprogram alih-alih menggunakan. AWS CLI

Tutorial ini menunjukkan kepada Anda sebagian kecil dari fungsionalitas SSO di. AWS SDK untuk .NET Untuk detail selengkapnya tentang penggunaan IAM Identity Center dengan AWS SDK untuk .NET, lihat topik dengan informasi latar belakang. Dalam topik itu, lihat terutama deskripsi tingkat tinggi untuk skenario ini di subbagian yang disebut. Hanya aplikasi.NET

> **Note**
>
> Beberapa langkah dalam tutorial ini membantu Anda mengkonfigurasi layanan seperti AWS Organizations dan IAM Identity Center. Jika Anda sudah melakukan konfigurasi itu, atau jika Anda hanya tertarik pada kode, Anda dapat melompat ke bagian dengan kode contoh.

Prasyarat

- Konfigurasikan lingkungan pengembangan Anda jika Anda belum melakukannya. Ini dijelaskan dalam bagian seperti Menginstal dan mengonfigurasi toolchain Anda untuk AWS SDK untuk .NET danMemulai.
- Identifikasi atau buat setidaknya satu Akun AWS yang dapat Anda gunakan untuk menguji SSO. Untuk keperluan tutorial ini, ini disebut test Akun AWS atau hanya test account.
- Identifikasi pengguna SSO yang dapat menguji SSO untuk Anda. Ini adalah orang yang akan menggunakan SSO dan aplikasi dasar yang Anda buat. Untuk tutorial ini, orang itu mungkin Anda (pengembang), atau orang lain. Kami juga merekomendasikan pengaturan di mana pengguna SSO bekerja pada komputer yang tidak ada di lingkungan pengembangan Anda. Namun, ini tidak sepenuhnya diperlukan.
- Komputer pengguna SSO harus memiliki framework .NET yang diinstal yang kompatibel dengan yang Anda gunakan untuk mengatur lingkungan pengembangan Anda.

Mengatur AWS

Bagian ini menunjukkan kepada Anda cara mengatur berbagai AWS layanan untuk tutorial ini.

Untuk melakukan pengaturan ini, pertama-tama masuk ke pengujian Akun AWS sebagai administrator. Kemudian, lakukan hal berikut:

Amazon S3

Buka konsol Amazon S3 dan tambahkan beberapa ember yang tidak berbahaya. Kemudian dalam tutorial ini, pengguna SSO akan mengambil daftar bucket ini.

AWS IAM

Buka konsol IAM dan tambahkan beberapa pengguna IAM. Jika Anda memberikan izin kepada pengguna IAM, batasi izin ke beberapa izin hanya-baca yang tidak berbahaya. Kemudian dalam tutorial ini, pengguna SSO akan mengambil daftar pengguna IAM ini.

AWS Organizations

Buka AWS Organizations konsol dan aktifkan Organizations. Untuk informasi selengkapnya, lihat Membuat organisasi di Panduan AWS Organizations Pengguna.

Tindakan ini menambahkan tes Akun AWS ke organisasi sebagai akun manajemen. Jika Anda memiliki akun pengujian tambahan, Anda dapat mengundang mereka untuk bergabung dengan organisasi, tetapi melakukannya tidak diperlukan untuk tutorial ini.

Pusat Identitas IAM

Buka konsol Pusat Identitas IAM dan aktifkan SSO. Lakukan verifikasi email jika perlu. Untuk informasi selengkapnya, lihat Mengaktifkan Pusat Identitas IAM di Panduan Pengguna Pusat Identitas IAM.

Kemudian, lakukan konfigurasi berikut.

Konfigurasikan Pusat Identitas IAM

1. Buka halaman Pengaturan. Cari “URL portal akses” dan catat nilai untuk digunakan nanti dalam `sso_start_url` pengaturan.
2. Di spanduk Konsol Manajemen AWS, cari Wilayah AWS yang disetel saat Anda mengaktifkan SSO. Ini adalah menu dropdown di sebelah kiri ID. Akun AWS Rekam kode Wilayah untuk digunakan nanti dalam `sso_region` pengaturan. Kode ini akan mirip dengan`us-east-1`.
3. Buat pengguna SSO sebagai berikut:
    a. Buka halaman Pengguna.
    b. Pilih Tambahkan pengguna dan masukkan Nama Pengguna, Alamat email, Nama depan, dan Nama belakang pengguna. Lalu, pilih Selanjutnya.
    c. Pilih Berikutnya pada halaman untuk grup, lalu tinjau informasinya dan pilih Tambah pengguna.
4. Buat grup sebagai berikut:
    a. Buka halaman Grup.

b. Pilih Buat grup dan masukkan nama Grup dan Deskripsi grup.

c. Di bagian Tambahkan pengguna ke grup, pilih pengguna SSO uji yang Anda buat sebelumnya. Kemudian, pilih Buat grup.

5. Buat set izin sebagai berikut:

a. Buka halaman Set izin dan pilih Buat set izin.

b. Di bawah Jenis set izin, pilih Set izin khusus dan pilih Berikutnya.

c. Buka kebijakan Inline dan masukkan kebijakan berikut:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

d. Untuk tutorial ini, masukkan `SSOReadOnlyRole` sebagai nama set Izin. Tambahkan Deskripsi jika Anda mau dan kemudian pilih Berikutnya.

e. Tinjau informasi dan kemudian pilih Buat.

f. Catat nama set izin untuk digunakan nanti dalam `sso_role_name` pengaturan.

6. Buka halaman AWS akun dan pilih AWS akun yang Anda tambahkan ke organisasi sebelumnya.

7. Di bagian Ikhtisar halaman itu, temukan ID Akun dan rekam untuk digunakan nanti dalam `sso_account_id` pengaturan.

8. Pilih tab Pengguna dan grup, lalu pilih Tetapkan pengguna atau grup.

9. Pada halaman Tetapkan pengguna dan grup, pilih tab Grup, pilih grup yang Anda buat sebelumnya, dan pilih Berikutnya.

10. Pilih set izin yang Anda buat sebelumnya dan pilih Berikutnya, lalu pilih Kirim. Konfigurasi memakan waktu beberapa saat.

Buat contoh aplikasi

Buat aplikasi berikut. Mereka akan dijalankan di komputer pengguna SSO.

Daftar ember Amazon S3

Sertakan NuGet paket `AWSSDK.SSO` dan `AWSSDK.SSOOIDC` sebagai tambahan untuk `AWSSDK.S3` dan`AWSSDK.SecurityToken`.

```
using System;
using System.Threading.Tasks;
using System.Diagnostics;


// NuGet packages: AWSSDK.S3, AWSSDK.SecurityToken, AWSSDK.SSO, AWSSDK.SSOOIDC
using Amazon.Runtime;
using Amazon.Runtime.CredentialManagement;
using Amazon.S3;
using Amazon.S3.Model;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;

namespace SSOExample.S3.Programmatic_login
{
    class Program
    {
        // Requirements:
        // - An SSO profile in the SSO user's shared config file.

        // Class members.
        private static string profile = "my-sso-profile";

        static async Task Main(string[] args)
        {
            // Get SSO credentials from the information in the shared config file.
            var ssoCreds = LoadSsoCredentials(profile);

            // Display the caller's identity.
            var ssoProfileClient = new AmazonSecurityTokenServiceClient(ssoCreds);
```

```
            Console.WriteLine($"\nSSO Profile:\n {await
ssoProfileClient.GetCallerIdentityArn()}");

            // Display a list of the account's S3 buckets.
            // The S3 client is created using the SSO credentials obtained earlier.
            var s3Client = new AmazonS3Client(ssoCreds);
            Console.WriteLine("\nGetting a list of your buckets...");
            var listResponse = await s3Client.ListBucketsAsync();
            Console.WriteLine($"Number of buckets: {listResponse.Buckets.Count}");
            foreach (S3Bucket b in listResponse.Buckets)
            {
                Console.WriteLine(b.BucketName);
            }
            Console.WriteLine();
        }

        // Method to get SSO credentials from the information in the shared config
file.
        static AWSCredentials LoadSsoCredentials(string profile)
        {
            var chain = new CredentialProfileStoreChain();
            if (!chain.TryGetAWSCredentials(profile, out var credentials))
                throw new Exception($"Failed to find the {profile} profile");

            var ssoCredentials = credentials as SSOAWSCredentials;

            ssoCredentials.Options.ClientName = "Example-SSO-App";
            ssoCredentials.Options.SsoVerificationCallback = args =>
            {
                // Launch a browser window that prompts the SSO user to complete an SSO
login.
                //  This method is only invoked if the session doesn't already have a
valid SSO token.
                // NOTE: Process.Start might not support launching a browser on macOS
or Linux. If not,
                //       use an appropriate mechanism on those systems instead.
                Process.Start(new ProcessStartInfo
                {
                    FileName = args.VerificationUriComplete,
                    UseShellExecute = true
                });
            };
            ssoCredentials.Options.SupportsGettingNewToken = true;
```

```
            return ssoCredentials;
        }

    }

    // Class to read the caller's identity.
    public static class Extensions
    {
        public static async Task<string> GetCallerIdentityArn(this
 IAmazonSecurityTokenService stsClient)
        {
            var response = await stsClient.GetCallerIdentityAsync(new
 GetCallerIdentityRequest());
            return response.Arn;
        }
    }
}
```

Daftar pengguna IAM

Sertakan NuGet paket `AWSSDK.SSO` dan `AWSSDK.SSOOIDC` sebagai tambahan untuk `AWSSDK.IdentityManagement` dan`AWSSDK.SecurityToken`.

```
using System;
using System.Threading.Tasks;
using System.Diagnostics;


// NuGet packages: AWSSDK.IdentityManagement, AWSSDK.SecurityToken, AWSSDK.SSO,
 AWSSDK.SSOOIDC
using Amazon.Runtime;
using Amazon.Runtime.CredentialManagement;
using Amazon.IdentityManagement;
using Amazon.IdentityManagement.Model;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;

namespace SSOExample.IAM.Programmatic_login
{
    class Program
    {
        // Requirements:
        // - An SSO profile in the SSO user's shared config file.
```

```
        // Class members.
        private static string profile = "my-sso-profile";

        static async Task Main(string[] args)
        {
            // Get SSO credentials from the information in the shared config file.
            var ssoCreds = LoadSsoCredentials(profile);

            // Display the caller's identity.
            var ssoProfileClient = new AmazonSecurityTokenServiceClient(ssoCreds);
            Console.WriteLine($"\nSSO Profile:\n {await
ssoProfileClient.GetCallerIdentityArn()}");

            // Display a list of the account's IAM users.
            // The IAM client is created using the SSO credentials obtained earlier.
            var iamClient = new AmazonIdentityManagementServiceClient(ssoCreds);
            Console.WriteLine("\nGetting a list of IAM users...");
            var listResponse = await iamClient.ListUsersAsync();
            Console.WriteLine($"Number of IAM users: {listResponse.Users.Count}");
            foreach (User u in listResponse.Users)
            {
                Console.WriteLine(u.UserName);
            }
            Console.WriteLine();
        }

        // Method to get SSO credentials from the information in the shared config
file.
        static AWSCredentials LoadSsoCredentials(string profile)
        {
            var chain = new CredentialProfileStoreChain();
            if (!chain.TryGetAWSCredentials(profile, out var credentials))
                throw new Exception($"Failed to find the {profile} profile");

            var ssoCredentials = credentials as SSOAWSCredentials;

            ssoCredentials.Options.ClientName = "Example-SSO-App";
            ssoCredentials.Options.SsoVerificationCallback = args =>
            {
                // Launch a browser window that prompts the SSO user to complete an SSO
login.
                //  This method is only invoked if the session doesn't already have a
valid SSO token.
```

```
                // NOTE: Process.Start might not support launching a browser on macOS
 or Linux. If not,
                //       use an appropriate mechanism on those systems instead.
                Process.Start(new ProcessStartInfo
                {
                    FileName = args.VerificationUriComplete,
                    UseShellExecute = true
                });
            };
            ssoCredentials.Options.SupportsGettingNewToken = true;

            return ssoCredentials;
        }

    }

    // Class to read the caller's identity.
    public static class Extensions
    {
        public static async Task<string> GetCallerIdentityArn(this
 IAmazonSecurityTokenService stsClient)
        {
            var response = await stsClient.GetCallerIdentityAsync(new
 GetCallerIdentityRequest());
            return response.Arn;
        }
    }
}
```

Selain menampilkan daftar bucket Amazon S3 dan pengguna IAM, aplikasi ini menampilkan identitas pengguna ARN untuk profil berkemampuan SSO, yang ada dalam tutorial ini. `my-sso-profile`

Aplikasi ini melakukan tugas masuk SSO dengan menyediakan metode callback di properti Options dari suatu objek. SSOAWSCredentials

Instruksikan pengguna SSO

Mintalah pengguna SSO untuk memeriksa email mereka dan menerima undangan SSO. Mereka diminta untuk mengatur kata sandi. Pesan mungkin membutuhkan waktu beberapa menit untuk tiba di kotak masuk pengguna SSO.

Berikan pengguna SSO aplikasi yang Anda buat sebelumnya.

Kemudian, mintalah pengguna SSO melakukan hal berikut:

1. Jika folder yang berisi AWS `config` file bersama tidak ada, buatlah. Jika folder memang ada dan memiliki subfolder bernama `.sso`, hapus subfolder itu.

   Lokasi folder ini biasanya `%USERPROFILE%\.aws` di Windows dan `~/.aws` di Linux dan macOS.

2. Buat AWS `config` file bersama di folder itu, jika perlu, dan tambahkan profil ke dalamnya sebagai berikut:

   ```
   [default]
   region = <default Region>

   [profile my-sso-profile]
   sso_start_url = <user portal URL recorded earlier>
   sso_region = <Region code recorded earlier>
   sso_account_id = <account ID recorded earlier>
   sso_role_name = SSOReadOnlyRole
   ```

3. Jalankan aplikasi Amazon S3.
4. Di halaman login web yang dihasilkan, masuk. Gunakan nama pengguna dari pesan undangan dan kata sandi yang dibuat sebagai respons terhadap pesan tersebut.
5. Saat login selesai, aplikasi menampilkan daftar bucket S3.
6. Jalankan aplikasi IAM. Aplikasi ini menampilkan daftar pengguna IAM. Ini benar meskipun login kedua tidak dilakukan. Aplikasi IAM menggunakan token sementara yang dibuat sebelumnya.

Pembersihan

Jika Anda tidak ingin menyimpan sumber daya yang Anda buat selama tutorial ini, bersihkan. Ini mungkin AWS sumber daya atau sumber daya di lingkungan pengembangan Anda seperti file dan folder.

Tutorial untuk SSO menggunakan aplikasi AWS CLI dan .NET

Tutorial ini menunjukkan cara mengaktifkan SSO untuk aplikasi .NET dasar dan pengguna SSO uji. Ini menggunakan AWS CLI untuk menghasilkan token SSO sementara alih-alih menghasilkannya secara terprogram.

Tutorial ini menunjukkan kepada Anda sebagian kecil dari fungsionalitas SSO di. AWS SDK untuk .NET Untuk detail lengkap tentang menggunakan IAM Identity Center dengan AWS SDK untuk .NET, lihat topik dengan informasi latar belakang. Dalam topik itu, lihat terutama deskripsi tingkat tinggi untuk skenario ini di subbagian yang disebut. AWS CLI dan aplikasi.NET

> **Note**
>
> Beberapa langkah dalam tutorial ini membantu Anda mengkonfigurasi layanan seperti AWS Organizations dan IAM Identity Center. Jika Anda telah melakukan konfigurasi tersebut, atau jika Anda hanya tertarik pada kode, Anda dapat melompat ke bagian dengan kode contoh.

Prasyarat

- Konfigurasikan lingkungan pengembangan Anda jika Anda belum melakukannya. Ini dijelaskan dalam bagian seperti Menginstal dan mengonfigurasi toolchain Anda untuk AWS SDK untuk .NET dan Memulai.
- Identifikasi atau buat setidaknya satu Akun AWS yang dapat Anda gunakan untuk menguji SSO. Untuk keperluan tutorial ini, ini disebut test Akun AWS atau hanya test account.
- Identifikasi pengguna SSO yang dapat menguji SSO untuk Anda. Ini adalah orang yang akan menggunakan SSO dan aplikasi dasar yang Anda buat. Untuk tutorial ini, orang itu mungkin Anda (pengembang), atau orang lain. Kami juga merekomendasikan pengaturan di mana pengguna SSO bekerja pada komputer yang tidak ada di lingkungan pengembangan Anda. Namun, ini tidak sepenuhnya diperlukan.
- Komputer pengguna SSO harus memiliki framework .NET yang diinstal yang kompatibel dengan yang Anda gunakan untuk mengatur lingkungan pengembangan Anda.

- Pastikan bahwa AWS CLI versi 2 diinstal pada komputer pengguna SSO. Anda dapat memeriksa ini dengan menjalankan `aws --version` command prompt atau terminal.

Mengatur AWS

Bagian ini menunjukkan kepada Anda cara mengatur berbagai AWS layanan untuk tutorial ini.

Untuk melakukan pengaturan ini, pertama-tama masuk ke pengujian Akun AWS sebagai administrator. Kemudian, lakukan hal berikut:

Amazon S3

Buka konsol Amazon S3 dan tambahkan beberapa ember yang tidak berbahaya. Kemudian dalam tutorial ini, pengguna SSO akan mengambil daftar bucket ini.

AWS IAM

Buka konsol IAM dan tambahkan beberapa pengguna IAM. Jika Anda memberikan izin kepada pengguna IAM, batasi izin ke beberapa izin hanya-baca yang tidak berbahaya. Kemudian dalam tutorial ini, pengguna SSO akan mengambil daftar pengguna IAM ini.

AWS Organizations

Buka AWS Organizations konsol dan aktifkan Organizations. Untuk informasi selengkapnya, lihat Membuat organisasi di Panduan AWS Organizations Pengguna.

Tindakan ini menambahkan tes Akun AWS ke organisasi sebagai akun manajemen. Jika Anda memiliki akun pengujian tambahan, Anda dapat mengundang mereka untuk bergabung dengan organisasi, tetapi melakukannya tidak diperlukan untuk tutorial ini.

Pusat Identitas IAM

Buka konsol Pusat Identitas IAM dan aktifkan SSO. Lakukan verifikasi email jika perlu. Untuk informasi selengkapnya, lihat Mengaktifkan Pusat Identitas IAM di Panduan Pengguna Pusat Identitas IAM.

Kemudian, lakukan konfigurasi berikut.

Konfigurasikan Pusat Identitas IAM

1. Buka halaman Pengaturan. Cari “URL portal akses” dan catat nilai untuk digunakan nanti dalam `sso_start_url` pengaturan.
2. Di spanduk Konsol Manajemen AWS, cari Wilayah AWS yang disetel saat Anda mengaktifkan SSO. Ini adalah menu dropdown di sebelah kiri ID. Akun AWS Rekam kode Wilayah untuk digunakan nanti dalam `sso_region` pengaturan. Kode ini akan mirip dengan`us-east-1`.
3. Buat pengguna SSO sebagai berikut:

    a. Buka halaman Pengguna.

    b. Pilih Tambahkan pengguna dan masukkan Nama Pengguna, Alamat email, Nama depan, dan Nama belakang pengguna. Lalu, pilih Selanjutnya.

c. Pilih Berikutnya pada halaman untuk grup, lalu tinjau informasinya dan pilih Tambah pengguna.

4. Buat grup sebagai berikut:

   a. Buka halaman Grup.

   b. Pilih Buat grup dan masukkan nama Grup dan Deskripsi grup.

   c. Di bagian Tambahkan pengguna ke grup, pilih pengguna SSO uji yang Anda buat sebelumnya. Kemudian, pilih Buat grup.

5. Buat set izin sebagai berikut:

   a. Buka halaman Set izin dan pilih Buat set izin.

   b. Di bawah Jenis set izin, pilih Set izin khusus dan pilih Berikutnya.

   c. Buka kebijakan Inline dan masukkan kebijakan berikut:

      JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

   d. Untuk tutorial ini, masukkan `SSOReadOnlyRole` sebagai nama set Izin. Tambahkan Deskripsi jika Anda mau dan kemudian pilih Berikutnya.

   e. Tinjau informasi dan kemudian pilih Buat.

   f. Catat nama set izin untuk digunakan nanti dalam `sso_role_name` pengaturan.

6. Buka halaman AWS akun dan pilih AWS akun yang Anda tambahkan ke organisasi sebelumnya.

7. Di bagian Ikhtisar halaman itu, temukan ID Akun dan rekam untuk digunakan nanti dalam `sso_account_id` pengaturan.
8. Pilih tab Pengguna dan grup, lalu pilih Tetapkan pengguna atau grup.
9. Pada halaman Tetapkan pengguna dan grup, pilih tab Grup, pilih grup yang Anda buat sebelumnya, dan pilih Berikutnya.
10. Pilih set izin yang Anda buat sebelumnya dan pilih Berikutnya, lalu pilih Kirim. Konfigurasi memakan waktu beberapa saat.

Buat contoh aplikasi

Buat aplikasi berikut. Mereka akan dijalankan di komputer pengguna SSO.

Daftar ember Amazon S3

Sertakan NuGet paket `AWSSDK.SSO` dan `AWSSDK.SSOOIDC` selain `AWSSDK.S3` dan`AWSSDK.SecurityToken`.

```
using System;
using System.Threading.Tasks;

// NuGet packages: AWSSDK.S3, AWSSDK.SecurityToken, AWSSDK.SSO, AWSSDK.SSOOIDC
using Amazon.Runtime;
using Amazon.Runtime.CredentialManagement;
using Amazon.S3;
using Amazon.S3.Model;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;

namespace SSOExample.S3.CLI_login
{
    class Program
    {
        // Requirements:
        // - An SSO profile in the SSO user's shared config file.
        // - An active SSO Token.
        //    If an active SSO token isn't available, the SSO user should do the
 following:
        //    In a terminal, the SSO user must call "aws sso login --profile my-sso-
profile".

        // Class members.
```

```
        private static string profile = "my-sso-profile";
        static async Task Main(string[] args)
        {
            // Get SSO credentials from the information in the shared config file.
            var ssoCreds = LoadSsoCredentials(profile);

            // Display the caller's identity.
            var ssoProfileClient = new AmazonSecurityTokenServiceClient(ssoCreds);
            Console.WriteLine($"\nSSO Profile:\n {await
ssoProfileClient.GetCallerIdentityArn()}");

            // Display a list of the account's S3 buckets.
            // The S3 client is created using the SSO credentials obtained earlier.
            var s3Client = new AmazonS3Client(ssoCreds);
            Console.WriteLine("\nGetting a list of your buckets...");
            var listResponse = await s3Client.ListBucketsAsync();
            Console.WriteLine($"Number of buckets: {listResponse.Buckets.Count}");
            foreach (S3Bucket b in listResponse.Buckets)
            {
                Console.WriteLine(b.BucketName);
            }
            Console.WriteLine();
        }

        // Method to get SSO credentials from the information in the shared config
file.
        static AWSCredentials LoadSsoCredentials(string profile)
        {
            var chain = new CredentialProfileStoreChain();
            if (!chain.TryGetAWSCredentials(profile, out var credentials))
                throw new Exception($"Failed to find the {profile} profile");
            return credentials;
        }
    }

    // Class to read the caller's identity.
    public static class Extensions
    {
        public static async Task<string> GetCallerIdentityArn(this
IAmazonSecurityTokenService stsClient)
        {
            var response = await stsClient.GetCallerIdentityAsync(new
GetCallerIdentityRequest());
            return response.Arn;
```

```
        }
    }
}
```

Daftar pengguna IAM

Sertakan NuGet paket `AWSSDK.SSO` dan `AWSSDK.SSOOIDC` selain `AWSSDK.IdentityManagement` dan`AWSSDK.SecurityToken`.

```
using System;
using System.Threading.Tasks;

// NuGet packages: AWSSDK.IdentityManagement, AWSSDK.SecurityToken, AWSSDK.SSO,
 AWSSDK.SSOOIDC
using Amazon.Runtime;
using Amazon.Runtime.CredentialManagement;
using Amazon.IdentityManagement;
using Amazon.IdentityManagement.Model;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;

namespace SSOExample.IAM.CLI_login
{
    class Program
    {
        // Requirements:
        // - An SSO profile in the SSO user's shared config file.
        // - An active SSO Token.
        //    If an active SSO token isn't available, the SSO user should do the
 following:
        //    In a terminal, the SSO user must call "aws sso login --profile my-sso-
profile".

        // Class members.
        private static string profile = "my-sso-profile";
        static async Task Main(string[] args)
        {
            // Get SSO credentials from the information in the shared config file.
            var ssoCreds = LoadSsoCredentials(profile);

            // Display the caller's identity.
            var ssoProfileClient = new AmazonSecurityTokenServiceClient(ssoCreds);
```

```
            Console.WriteLine($"\nSSO Profile:\n {await
 ssoProfileClient.GetCallerIdentityArn()}");

            // Display a list of the account's IAM users.
            // The IAM client is created using the SSO credentials obtained earlier.
            var iamClient = new AmazonIdentityManagementServiceClient(ssoCreds);
            Console.WriteLine("\nGetting a list of IAM users...");
            var listResponse = await iamClient.ListUsersAsync();
            Console.WriteLine($"Number of IAM users: {listResponse.Users.Count}");
            foreach (User u in listResponse.Users)
            {
                Console.WriteLine(u.UserName);
            }
            Console.WriteLine();
        }

        // Method to get SSO credentials from the information in the shared config
 file.
        static AWSCredentials LoadSsoCredentials(string profile)
        {
            var chain = new CredentialProfileStoreChain();
            if (!chain.TryGetAWSCredentials(profile, out var credentials))
                throw new Exception($"Failed to find the {profile} profile");
            return credentials;
        }
    }

    // Class to read the caller's identity.
    public static class Extensions
    {
        public static async Task<string> GetCallerIdentityArn(this
 IAmazonSecurityTokenService stsClient)
        {
            var response = await stsClient.GetCallerIdentityAsync(new
 GetCallerIdentityRequest());
            return response.Arn;
        }
    }
}
```

Selain menampilkan daftar bucket Amazon S3 dan pengguna IAM, aplikasi ini menampilkan identitas pengguna ARN untuk profil berkemampuan SSO, yang ada dalam tutorial ini. `my-sso-profile`

Instruksikan pengguna SSO

Mintalah pengguna SSO untuk memeriksa email mereka dan menerima undangan SSO. Mereka diminta untuk mengatur kata sandi. Pesan mungkin membutuhkan waktu beberapa menit untuk tiba di kotak masuk pengguna SSO.

Berikan pengguna SSO aplikasi yang Anda buat sebelumnya.

Kemudian, mintalah pengguna SSO melakukan hal berikut:

1. Jika folder yang berisi AWS `config` file bersama tidak ada, buatlah. Jika folder memang ada dan memiliki subfolder bernama `.sso`, hapus subfolder itu.

   Lokasi folder ini biasanya `%USERPROFILE%\.aws` di Windows dan `~/.aws` di Linux dan macOS.

2. Buat AWS `config` file bersama di folder itu, jika perlu, dan tambahkan profil ke dalamnya sebagai berikut:

   ```
   [default]
   region = <default Region>

   [profile my-sso-profile]
   sso_start_url = <user portal URL recorded earlier>
   sso_region = <Region code recorded earlier>
   sso_account_id = <account ID recorded earlier>
   sso_role_name = SSOReadOnlyRole
   ```

3. Jalankan aplikasi Amazon S3. Pengecualian runtime muncul.
4. Jalankan AWS CLI perintah berikut:

   ```
   aws sso login --profile my-sso-profile
   ```

5. Di halaman login web yang dihasilkan, masuk. Gunakan nama pengguna dari pesan undangan dan kata sandi yang dibuat sebagai respons terhadap pesan tersebut.
6. Jalankan aplikasi Amazon S3 lagi. Aplikasi sekarang menampilkan daftar ember S3.
7. Jalankan aplikasi IAM. Aplikasi ini menampilkan daftar pengguna IAM. Ini benar meskipun login kedua tidak dilakukan. Aplikasi IAM menggunakan token sementara yang dibuat sebelumnya.

Pembersihan

Jika Anda tidak ingin menyimpan sumber daya yang Anda buat selama tutorial ini, bersihkan. Ini mungkin AWS sumber daya atau sumber daya di lingkungan pengembangan Anda seperti file dan folder.

Menyebarkan aplikasi ke AWS

Setelah mengembangkan aplikasi atau layanan .NET Core cloud-native di mesin pengembangan, Anda akan ingin menerapkannya. AWS Anda dapat melakukan ini dengan menggunakan Konsol Manajemen AWS atau layanan tertentu seperti CloudFormation atau AWS Cloud Development Kit (AWS CDK). Anda juga dapat menggunakan AWS alat yang telah dibuat untuk tujuan penyebaran. Dengan menggunakan alat-alat ini, Anda dapat melakukan hal berikut.

Terapkan dari .NET CLI

Anda dapat menggunakan AWS alat-alat berikut untuk.NET CLI untuk menyebarkan aplikasi Anda ke: AWS

- AWS Menyebarkan Alat untuk.NET CLI - Mendukung penerapan AWS App Runnerke, Amazon Elastic Container Service (Amazon ECS), dan. AWS Elastic Beanstalk
- AWS Lambda Alat untuk.NET CLI - Mendukung penyebaran proyek. AWS Lambda

Terapkan dari toolkit IDE

Anda dapat menggunakan AWS toolkit untuk menyebarkan aplikasi Anda langsung dari IDE pilihan Anda:

- AWS Toolkit for Visual Studio

  > **Note**
  >
  > Fitur “Publish to AWS” di toolkit memperlihatkan fungsionalitas yang sama dengan AWS Deploy Tool untuk.NET CLI. Untuk mempelajari lebih lanjut, buka Publikasikan ke AWS dalam Panduan AWS Toolkit for Visual Studio Pengguna.

- AWS Toolkit for JetBrains

  Lihat Bekerja dengan Aplikasi AWS Tanpa Server dan Bekerja dengan. AWS App Runner

- AWS Toolkit for VS Code

  Lihat Bekerja dengan aplikasi tanpa server dan Menggunakan. AWS App Runner

- AWS Toolkit for Azure DevOps

Kasus penggunaan

Bagian berikut berisi skenario kasus penggunaan untuk jenis aplikasi tertentu, termasuk informasi tentang bagaimana Anda akan menggunakan .NET CLI untuk menyebarkan aplikasi tersebut.

- Aplikasi ASP.NET Core
- Aplikasi Konsol .NET
- Aplikasi Blazor WebAssembly
- AWS Lambda proyek

Aplikasi ASP.NET Core

Alat AWS Deploy untuk .NET CLI membantu Anda menyebarkan aplikasi ASP.NET Anda dan memandu Anda melalui proses penyebaran. Ini adalah alat interaktif untuk CLI .NET yang membantu menyebarkan aplikasi.NET dengan pengetahuan minimum. AWS

Alat Deploy memiliki kemampuan sebagai berikut:

- Rekomendasi komputasi untuk aplikasi Anda - Dapatkan rekomendasi komputasi dan pelajari AWS komputasi mana yang paling cocok untuk aplikasi Anda.
- Generasi Dockerfile - Alat ini menghasilkan Dockerfile jika diperlukan, atau menggunakan Dockerfile yang ada.
- Pengemasan dan penerapan otomatis — Alat ini membangun artefak penerapan, menyediakan infrastruktur dengan menggunakan proyek AWS CDK penerapan yang dihasilkan, dan menyebarkan aplikasi Anda ke komputasi yang dipilih. AWS
- Penerapan berulang dan dapat dibagikan — Anda dapat membuat dan memodifikasi proyek AWS CDK penerapan agar sesuai dengan kasus penggunaan spesifik Anda. Anda juga dapat mengontrol versi proyek Anda dan membagikannya dengan tim Anda untuk penerapan berulang.
- Help with learning AWS CDK for .NET - Alat ini membantu Anda secara bertahap mempelajari AWS alat-alat yang mendasarinya, seperti AWS CDK.

Alat AWS Deploy mendukung penerapan aplikasi ASP.NET Core ke layanan berikut: AWS

- Layanan Amazon ECS menggunakan AWS Fargate- Mendukung penerapan aplikasi web ke Amazon Elastic Container Service (Amazon ECS) dengan daya komputasi yang dikelola oleh mesin komputasi tanpa server. AWS Fargate

- AWS App Runner- Mendukung penerapan ke layanan yang dikelola sepenuhnya yang memudahkan pengembang untuk menyebarkan aplikasi web kontainer dan dalam skala besar. APIs Tidak diperlukan pengalaman infrastruktur sebelumnya.
- AWS Elastic Beanstalk- Mendukung penerapan ke layanan yang memudahkan pengembang untuk menyebarkan aplikasi web dan APIs ke lingkungan yang dikelola sepenuhnya dalam skala besar. Tidak diperlukan pengalaman infrastruktur sebelumnya.

Untuk mempelajari lebih lanjut, lihat ikhtisar alat. Untuk memulai dari sana, navigasikan ke Dokumentasi, Memulai, dan pilih Cara menginstal untuk petunjuk instalasi.

Aplikasi Konsol .NET

Alat AWS Deploy untuk .NET CLI membantu Anda menyebarkan aplikasi.NET Console Anda sebagai layanan atau tugas terjadwal sebagai gambar kontainer di Linux dan memandu Anda melalui proses penyebaran. Jika aplikasi Anda tidak memiliki Dockerfile, alat secara otomatis menghasilkannya. Jika tidak, Dockerfile yang ada digunakan.

Alat Deploy memiliki kemampuan sebagai berikut:

- Rekomendasi komputasi untuk aplikasi Anda - Dapatkan rekomendasi komputasi dan pelajari AWS komputasi mana yang paling cocok untuk aplikasi Anda.
- Generasi Dockerfile - Alat ini menghasilkan Dockerfile jika diperlukan, atau menggunakan Dockerfile yang ada.
- Pengemasan dan penerapan otomatis — Alat ini membangun artefak penerapan, menyediakan infrastruktur dengan menggunakan proyek AWS CDK penerapan yang dihasilkan, dan menyebarkan aplikasi Anda ke komputasi yang dipilih. AWS
- Penerapan berulang dan dapat dibagikan — Anda dapat membuat dan memodifikasi proyek AWS CDK penerapan agar sesuai dengan kasus penggunaan spesifik Anda. Anda juga dapat mengontrol versi proyek Anda dan membagikannya dengan tim Anda untuk penerapan berulang.
- Help with learning AWS CDK for .NET - Alat ini membantu Anda secara bertahap mempelajari AWS alat-alat yang mendasarinya, seperti AWS CDK.

Alat AWS Deploy mendukung penerapan aplikasi.NET Console ke layanan berikut: AWS

- Amazon ECS Service menggunakan AWS Fargate- Mendukung penerapan aplikasi.NET sebagai layanan (misalnya, prosesor latar belakang) ke Amazon Elastic Container Service (Amazon

ECS) Service Elastic Container (Amazon ECS) dengan daya komputasi yang dikelola oleh mesin komputasi tanpa server. AWS Fargate

- Menggunakan Tugas Terjadwal Amazon ECS AWS Fargate- Mendukung penerapan aplikasi.NET sebagai tugas terjadwal (misalnya, end-of-day proses) ke Amazon ECS dengan daya komputasi yang dikelola oleh mesin komputasi tanpa server. AWS Fargate

Untuk mempelajari lebih lanjut, lihat ikhtisar alat. Untuk memulai dari sana, navigasikan ke Dokumentasi, Memulai, dan pilih Cara menginstal untuk petunjuk instalasi.

Aplikasi Blazor WebAssembly

Alat AWS Penyebaran untuk.NET CLI membantu Anda meng-host aplikasi WebAssembly Blazor Anda di Amazon S3, menggunakan Amazon untuk pengiriman jaringan konten. CloudFront Aplikasi Anda di-deploy ke bucket S3 untuk hosting web. Alat ini membuat dan mengonfigurasi bucket S3, lalu mengunggah aplikasi Blazor Anda ke bucket.

Alat Deploy memiliki kemampuan sebagai berikut:

- Pengemasan dan penerapan otomatis — Alat ini membangun artefak penerapan, menyediakan infrastruktur dengan menggunakan proyek AWS CDK penerapan yang dihasilkan, dan menyebarkan aplikasi Anda ke komputasi yang dipilih. AWS
- Penerapan berulang dan dapat dibagikan — Anda dapat membuat dan memodifikasi proyek AWS CDK penerapan agar sesuai dengan kasus penggunaan spesifik Anda. Anda juga dapat mengontrol versi proyek Anda dan membagikannya dengan tim Anda untuk penerapan berulang.
- Help with learning AWS CDK for .NET - Alat ini membantu Anda secara bertahap mempelajari AWS alat-alat yang mendasarinya, seperti AWS CDK.

Untuk mempelajari lebih lanjut, lihat ikhtisar alat. Untuk memulai dari sana, navigasikan ke Dokumentasi, Memulai, dan pilih Cara menginstal untuk petunjuk instalasi.

AWS Lambda proyek

AWS Lambda adalah layanan komputasi yang memungkinkan Anda menjalankan kode tanpa menyediakan atau mengelola server. Ini menjalankan kode Anda pada infrastruktur komputasi ketersediaan tinggi dan melakukan semua administrasi sumber daya komputasi. Untuk informasi lebih lanjut tentang Lambda, lihat Apa itu AWS Lambda? di Panduan AWS Lambda Pengembang.

Anda dapat menggunakan fungsi Lambda dengan menggunakan antarmuka baris perintah .NET (CLI).

Topik

- Prasyarat
- Perintah Lambda yang tersedia
- Langkah-langkah untuk menyebarkan

Prasyarat

Sebelum Anda mulai menggunakan .NET CLI untuk menyebarkan fungsi Lambda, Anda harus memenuhi prasyarat berikut:

- Konfirmasikan bahwa Anda telah menginstal .NET CLI. Sebagai contoh: `dotnet --version`. Jika perlu, buka https://dotnet.microsoft.com/download untuk menginstalnya.
- Siapkan CLI .NET untuk bekerja dengan Lambda. Untuk penjelasan tentang cara melakukannya, lihat .NET Core CLI di Panduan AWS Lambda Pengembang. Dalam prosedur itu, berikut ini adalah perintah penerapan:

  ```
  dotnet lambda deploy-function MyFunction --function-role role
  ```

  Jika Anda tidak yakin bagaimana membuat peran IAM untuk latihan ini, jangan sertakan `--function-role role` bagiannya. Alat ini akan membantu Anda membuat peran baru.

Perintah Lambda yang tersedia

Untuk membuat daftar perintah Lambda yang tersedia melalui .NET CLI, buka prompt perintah atau terminal dan masukkan. `dotnet lambda --help` Output perintah akan mirip dengan yang berikut:

```
Amazon Lambda Tools for .NET applications
Project Home: https://github.com/aws/aws-extensions-for-dotnet-cli, https://github.com/
aws/aws-lambda-dotnet

Commands to deploy and manage AWS Lambda functions:

        deploy-function         Command to deploy the project to AWS Lambda
        ...
```

```
        (etc.)

To get help on individual commands execute:
        dotnet lambda help <command>
```

Output mencantumkan semua perintah yang saat ini tersedia.

Langkah-langkah untuk menyebarkan

Instruksi berikut mengasumsikan bahwa Anda telah membuat sebuah AWS Lambda proyek.NET. Untuk keperluan prosedur ini, proyek diberi nama`DotNetCoreLambdaTest`.

1. Buka prompt perintah atau terminal, dan arahkan ke folder yang berisi file proyek.NET Lambda Anda.
2. Masukkan `dotnet lambda deploy-function`.
3. Jika diminta, masukkan AWS Wilayah (Wilayah tempat fungsi Lambda Anda akan digunakan).
4. Saat diminta, masukkan nama fungsi yang akan digunakan, misalnya,. `DotNetCoreLambdaTest` Ini bisa berupa nama fungsi yang sudah ada di Anda Akun AWS atau yang belum digunakan di sana.
5. Saat diminta, pilih atau buat peran IAM yang akan diasumsikan Lambda saat menjalankan fungsi.

Setelah berhasil diselesaikan, pesan Fungsi Lambda Baru yang dibuat ditampilkan.

```
Executing publish command
...
(etc.)
New Lambda function created
```

Jika Anda menerapkan fungsi yang sudah ada di akun Anda, fungsi deploy hanya meminta AWS Wilayah (jika perlu). Dalam hal ini, output perintah diakhiri dengan`Updating code for existing function`.

Setelah fungsi Lambda Anda di-deploy, itu siap digunakan. Untuk informasi selengkapnya, lihat Contoh Cara Menggunakan AWS Lambda.

Lambda secara otomatis memonitor fungsi Lambda untuk Anda dan melaporkan metrik melalui Amazon. CloudWatch Untuk memantau dan memecahkan masalah fungsi Lambda Anda, lihat Memantau dan memecahkan masalah aplikasi Lambda.

Memanggil AWS layanan dari AWS SDK untuk .NET

Bagian berikut berisi contoh, tutorial, tugas, dan panduan yang menunjukkan cara menggunakan AWS SDK untuk .NET untuk bekerja dengan AWS layanan. Contoh dan tutorial ini bergantung pada API yang AWS SDK untuk .NET disediakan. Untuk melihat kelas dan metode apa yang tersedia di API, lihat Referensi AWS SDK untuk .NET API.

Jika Anda baru mengenal AWS SDK untuk .NET, Anda mungkin ingin memeriksa Membuat aplikasi sederhana topiknya terlebih dahulu. Ini memberi Anda pengantar SDK.

Anda dapat menemukan lebih banyak contoh kode di Repositori Contoh AWS Kode dan repositori awslabs di. GitHub

Sebelum Anda mulai, pastikan Anda telah mengatur lingkungan Anda dan mengkonfigurasi proyek Anda. Juga tinjau informasi diMenggunakan SDK .

Topik

- Contoh kode terpandu untuk AWS SDK untuk .NET
- Menggunakan AWS Lambda untuk layanan komputasi
- Pustaka dan kerangka kerja tingkat tinggi untuk AWS SDK untuk .NET
- Support untuk AWS layanan dan konfigurasi lainnya

Contoh kode terpandu untuk AWS SDK untuk .NET

Bagian berikut berisi contoh kode dan memberikan panduan untuk contoh. Mereka dapat membantu Anda mempelajari cara menggunakan AWS SDK untuk .NET untuk bekerja dengan AWS layanan. Untuk contoh kode tambahan, lihatContoh kode.

> **Note**
>
> Contoh kode yang khusus untuk V3 juga AWS SDK untuk .NET tersedia. Untuk menemukannya, lihat Contoh kode dengan panduan dan contoh Kode di Panduan Pengembang AWS SDK untuk .NET (V3). Namun, ketahuilah bahwa jika Anda menggunakan contoh kode khusus V3 dengan V4 SDK (versi terbaru), Anda mungkin perlu melakukan penyesuaian sesuai dengan informasi di. Migrasi ke versi 4

Jika Anda baru mengenal AWS SDK untuk .NET, Anda mungkin ingin memeriksa Membuat aplikasi sederhana topiknya terlebih dahulu. Ini memberi Anda pengantar SDK.

Sebelum Anda mulai, pastikan Anda telah mengatur lingkungan Anda dan mengkonfigurasi proyek Anda. Juga tinjau informasi diMenggunakan SDK .

Topik

- Mengakses CloudFormation dengan AWS SDK untuk .NET
- Mengautentikasi pengguna dengan Amazon Cognito
- Menggunakan database Amazon DynamoDB NoSQL
- Bekerja dengan Amazon EC2
- Mengakses AWS Identity and Access Management (IAM) dengan AWS SDK untuk .NET
- Menggunakan penyimpanan Internet Amazon Simple Storage Service
- Mengirim Pemberitahuan Dari Cloud Menggunakan Amazon Simple Notification Service
- Pesan menggunakan Amazon SQS

Mengakses CloudFormation dengan AWS SDK untuk .NET

AWS SDK untuk .NET Dukungan AWS CloudFormation, yang menciptakan dan menyediakan penyebaran AWS infrastruktur dapat diprediksi dan berulang kali.

APIs

AWS SDK untuk .NET Menyediakan APIs untuk CloudFormation klien. Ini APIs memungkinkan Anda untuk bekerja dengan CloudFormation fitur seperti template dan tumpukan. Bagian ini berisi sejumlah kecil contoh yang menunjukkan pola yang dapat Anda ikuti saat bekerja dengan ini APIs. Untuk melihat set lengkap APIs, lihat Referensi AWS SDK untuk .NET API (dan gulir ke “Amazon. CloudFormation“).

Yang AWS CloudFormation APIs disediakan oleh AWSSDK. CloudFormationpaket.

Prasyarat

Sebelum Anda mulai, pastikan Anda telah mengatur lingkungan Anda dan mengkonfigurasi proyek Anda. Juga tinjau informasi diMenggunakan SDK .

Topik

Topik

- Daftar AWS sumber daya menggunakan AWS CloudFormation

Daftar AWS sumber daya menggunakan AWS CloudFormation

Contoh ini menunjukkan kepada Anda cara menggunakan daftar sumber daya di CloudFormation tumpukan. AWS SDK untuk .NET Contoh menggunakan API tingkat rendah. Aplikasi tidak mengambil argumen, tetapi hanya mengumpulkan informasi untuk semua tumpukan yang dapat diakses oleh kredensi pengguna dan kemudian menampilkan informasi tentang tumpukan tersebut.

Referensi SDK

NuGet paket:

- AWSSDK.CloudFormation

Elemen pemrograman:

- Namespace Amazon. CloudFormation

  Kelas AmazonCloudFormationClient
- Namespace Amazon. CloudFormation.Model

  Kelas ICloudFormationPaginatorFactory. DescribeStacks

  Kelas DescribeStackResourcesRequest

  Kelas DescribeStackResourcesResponse

  Stack Kelas

  Kelas StackResource

  Tag Kelas

```
using Amazon.CloudFormation;
using Amazon.CloudFormation.Model;
using Amazon.Runtime;
```

```
namespace CloudFormationActions;

public static class HelloCloudFormation
{
    public static IAmazonCloudFormation _amazonCloudFormation;

    static async Task Main(string[] args)
    {
        // Create the CloudFormation client
        _amazonCloudFormation = new AmazonCloudFormationClient();
        Console.WriteLine($"\nIn Region:
 {_amazonCloudFormation.Config.RegionEndpoint}");

        // List the resources for each stack
        await ListResources();
    }

    /// <summary>
    /// Method to list stack resources and other information.
    /// </summary>
    /// <returns>True if successful.</returns>
    public static async Task<bool> ListResources()
    {
        try
        {
            Console.WriteLine("Getting CloudFormation stack information...");

            // Get all stacks using the stack paginator.
            var paginatorForDescribeStacks =
                _amazonCloudFormation.Paginators.DescribeStacks(
                    new DescribeStacksRequest());
            await foreach (Stack stack in paginatorForDescribeStacks.Stacks)
            {
                // Basic information for each stack

 Console.WriteLine("\n-----------------------------------------------");
                Console.WriteLine($"\nStack: {stack.StackName}");
                Console.WriteLine($"  Status: {stack.StackStatus.Value}");
                Console.WriteLine($"  Created: {stack.CreationTime}");

                // The tags of each stack (etc.)
                if (stack.Tags.Count > 0)
                {
```

```
                    Console.WriteLine("  Tags:");
                    foreach (Tag tag in stack.Tags)
                        Console.WriteLine($"    {tag.Key}, {tag.Value}");
                }

                // The resources of each stack
                DescribeStackResourcesResponse responseDescribeResources =
                    await _amazonCloudFormation.DescribeStackResourcesAsync(
                        new DescribeStackResourcesRequest
                        {
                            StackName = stack.StackName
                        });
                if (responseDescribeResources.StackResources.Count > 0)
                {
                    Console.WriteLine("  Resources:");
                    foreach (StackResource resource in responseDescribeResources
                                .StackResources)
                        Console.WriteLine(
                            $"    {resource.LogicalResourceId}:
{resource.ResourceStatus}");
                }
            }

            Console.WriteLine("\n---------------------------------------------");
            return true;
        }
        catch (AmazonCloudFormationException ex)
        {
            Console.WriteLine("Unable to get stack information:\n" + ex.Message);
            return false;
        }
        catch (AmazonServiceException ex)
        {
            if (ex.Message.Contains("Unable to get IAM security credentials"))
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine("If you are usnig SSO, be sure to install" +
                                  " the AWSSDK.SSO and AWSSDK.SSOOIDC packages.");
            }
            else
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine(ex.StackTrace);
            }
```

```
            return false;
        }
        catch (ArgumentNullException ex)
        {
            if (ex.Message.Contains("Options property cannot be empty: ClientName"))
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine("If you are using SSO, have you logged in?");
            }
            else
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine(ex.StackTrace);
            }

            return false;
        }
    }
```

Mengautentikasi pengguna dengan Amazon Cognito

> **Note**
>
> Informasi dalam topik ini khusus untuk proyek berdasarkan .NET Framework dan AWS SDK untuk .NET versi 3.3 dan sebelumnya.

Menggunakan Amazon Cognito Identity, Anda dapat membuat identitas unik untuk pengguna Anda dan mengautentikasi mereka untuk akses aman ke AWS sumber daya Anda seperti Amazon S3 atau Amazon DynamoDB. Amazon Cognito Identity mendukung penyedia identitas publik seperti Amazon, Facebook, Twitter/Digit, Google, atau penyedia yang kompatibel dengan OpenID Connect serta identitas yang tidak diautentikasi. Amazon Cognito juga mendukung identitas otentikasi pengembang, yang memungkinkan Anda mendaftarkan dan mengautentikasi pengguna menggunakan proses otentikasi backend Anda sendiri, sambil tetap menggunakan Amazon Cognito Sync untuk menyinkronkan data pengguna dan mengakses sumber daya. AWS

Untuk informasi selengkapnya tentang Amazon Cognito, lihat Panduan Pengembang Amazon Cognito.

Contoh kode berikut menunjukkan cara mudah menggunakan Identitas Amazon Cognito. Penyedia kredensialContoh menunjukkan cara membuat dan mengautentikasi identitas pengguna. CognitoAuthentication perpustakaan ekstensiContoh menunjukkan cara menggunakan pustaka CognitoAuthentication ekstensi untuk mengautentikasi kumpulan pengguna Amazon Cognito.

Topik

- Penyedia kredensi Amazon Cognito
- Contoh perpustakaan CognitoAuthentication ekstensi Amazon

Penyedia kredensi Amazon Cognito

> **Note**
>
> Informasi dalam topik ini khusus untuk proyek berdasarkan .NET Framework dan AWS SDK untuk .NET versi 3.3 dan sebelumnya.

`Amazon.CognitoIdentity.CognitoAWSCredentials`, ditemukan di AWSSDK. CognitoIdentity NuGetpackage, adalah objek kredensial yang menggunakan Amazon Cognito dan AWS Security Token Service AWS STS() untuk mengambil kredensional untuk melakukan panggilan. AWS

Langkah pertama dalam menyiapkan `CognitoAWSCredentials` adalah membuat “kumpulan identitas”. (Kumpulan identitas adalah penyimpan informasi identitas pengguna yang spesifik untuk akun Anda. Informasi tersebut dapat diambil kembali di seluruh platform klien, perangkat, dan sistem operasi, sehingga jika pengguna mulai menggunakan aplikasi di ponsel dan kemudian beralih ke tablet, informasi aplikasi yang bertahan masih tersedia untuk pengguna tersebut. Anda dapat membuat kumpulan identitas baru dari konsol Amazon Cognito. Jika Anda menggunakan konsol, itu juga akan memberi Anda informasi lain yang Anda butuhkan:

- Nomor akun Anda - Nomor 12 digit, seperti 123456789012, yang unik untuk akun Anda.
- Peran yang tidak diautentikasi ARN- Peran yang akan diasumsikan oleh pengguna yang tidak diautentikasi. Misalnya, peran ini dapat memberikan izin hanya-baca ke data Anda.
- Peran yang diautentikasi ARN- Peran yang akan diasumsikan oleh pengguna yang diautentikasi. Peran ini dapat memberikan izin yang lebih luas untuk data Anda.

Mengatur Cognito AWSCredentials

Contoh kode berikut menunjukkan cara mengatur`CognitoAWSCredentials`, yang kemudian dapat Anda gunakan untuk melakukan panggilan ke Amazon S3 sebagai pengguna yang tidak diautentikasi. Ini memungkinkan Anda melakukan panggilan hanya dengan jumlah minimum data yang diperlukan untuk mengautentikasi pengguna. Izin pengguna dikendalikan oleh peran, sehingga Anda dapat mengonfigurasi akses sesuai kebutuhan.

```
CognitoAWSCredentials credentials = new CognitoAWSCredentials(
    accountId,        // Account number
    identityPoolId,   // Identity pool ID
    unAuthRoleArn,    // Role for unauthenticated users
    null,             // Role for authenticated users, not set
    region);
using (var s3Client = new AmazonS3Client(credentials))
{
    s3Client.ListBuckets();
}
```

Gunakan AWS sebagai pengguna yang tidak diautentikasi

Contoh kode berikut menunjukkan bagaimana Anda dapat mulai menggunakan AWS sebagai pengguna yang tidak diautentikasi, kemudian mengautentikasi melalui Facebook dan memperbarui kredensialnya untuk menggunakan kredensi Facebook. Dengan menggunakan pendekatan ini, Anda dapat memberikan kemampuan yang berbeda kepada pengguna yang diautentikasi melalui peran yang diautentikasi. Misalnya, Anda mungkin memiliki aplikasi telepon yang memungkinkan pengguna untuk melihat konten secara anonim, tetapi memungkinkan mereka untuk memposting jika mereka masuk dengan satu atau lebih penyedia yang dikonfigurasi.

```
CognitoAWSCredentials credentials = new CognitoAWSCredentials(
    accountId, identityPoolId,
    unAuthRoleArn,    // Role for unauthenticated users
    authRoleArn,      // Role for authenticated users
    region);
using (var s3Client = new AmazonS3Client(credentials))
{
    // Initial use will be unauthenticated
    s3Client.ListBuckets();

    // Authenticate user through Facebook
    string facebookToken = GetFacebookAuthToken();
```

```
    // Add Facebook login to credentials. This clears the current AWS credentials
    // and retrieves new AWS credentials using the authenticated role.
    credentials.AddLogin("graph.facebook.com", facebookAccessToken);

    // This call is performed with the authenticated role and credentials
    s3Client.ListBuckets();
}
```

`CognitoAWSCredentials`Objek menyediakan lebih banyak fungsionalitas jika Anda menggunakannya dengan `AmazonCognitoSyncClient` yang merupakan bagian dari AWS SDK untuk .NET. Jika Anda menggunakan keduanya `AmazonCognitoSyncClient` dan`CognitoAWSCredentials`, Anda tidak perlu menentukan `IdentityId` properti `IdentityPoolId` dan saat melakukan panggilan dengan`AmazonCognitoSyncClient`. Properti ini secara otomatis diisi dari`CognitoAWSCredentials`. Contoh kode berikutnya menggambarkan hal ini, serta peristiwa yang memberi tahu Anda setiap kali ada perubahan`IdentityId`. `CognitoAWSCredentials` `IdentityId`Dapat berubah dalam beberapa kasus, seperti ketika mengubah dari pengguna yang tidak diautentikasi ke yang diautentikasi.

```
CognitoAWSCredentials credentials = GetCognitoAWSCredentials();

// Log identity changes
credentials.IdentityChangedEvent += (sender, args) =>
{
    Console.WriteLine("Identity changed: [{0}] => [{1}]", args.OldIdentityId,
 args.NewIdentityId);
};

using (var syncClient = new AmazonCognitoSyncClient(credentials))
{
    var result = syncClient.ListRecords(new ListRecordsRequest
    {
        DatasetName = datasetName
        // No need to specify these properties
        //IdentityId = "...",
        //IdentityPoolId = "..."
    });
}
```

Contoh perpustakaan CognitoAuthentication ekstensi Amazon

> **Note**
>
> Informasi dalam topik ini khusus untuk proyek berdasarkan .NET Framework dan AWS SDK untuk .NET versi 3.3 dan sebelumnya.

Pustaka CognitoAuthentication ekstensi, ditemukan di Amazon.Extensions. CognitoAuthentication NuGet paket, menyederhanakan proses otentikasi kumpulan pengguna Amazon Cognito untuk pengembang.NET Core dan Xamarin. Pustaka dibangun di atas API penyedia Identitas Amazon Cognito untuk membuat dan mengirim panggilan API otentikasi pengguna.

Menggunakan pustaka CognitoAuthentication ekstensi

Amazon Cognito memiliki beberapa built-in `AuthFlow` dan `ChallengeName` nilai untuk alur otentikasi standar untuk memvalidasi nama pengguna dan kata sandi melalui Secure Remote Password (SRP). Untuk informasi selengkapnya tentang alur autentikasi, lihat Alur Otentikasi Kumpulan Pengguna Amazon Cognito.

Contoh-contoh berikut memerlukan `using` pernyataan ini:

```
// Required for all examples
using System;
using Amazon;
using Amazon.CognitoIdentity;
using Amazon.CognitoIdentityProvider;
using Amazon.Extensions.CognitoAuthentication;
using Amazon.Runtime;
// Required for the GetS3BucketsAsync example
using Amazon.S3;
using Amazon.S3.Model;
```

Gunakan otentikasi dasar

Buat AmazonCognitoIdentityProviderClientmenggunakan Anonymous AWSCredentials, yang tidak memerlukan permintaan yang ditandatangani. Anda tidak perlu menyediakan wilayah, kode yang mendasarinya memanggil `FallbackRegionFactory.GetRegionEndpoint()` jika suatu wilayah tidak disediakan. Buat `CognitoUserPool` dan `CognitoUser` objek. Panggil

`StartWithSrpAuthAsync` metode dengan `InitiateSrpAuthRequest` yang berisi kata sandi pengguna.

```
public static async void GetCredsAsync()
{
    AmazonCognitoIdentityProviderClient provider =
        new AmazonCognitoIdentityProviderClient(new
 Amazon.Runtime.AnonymousAWSCredentials());
    CognitoUserPool userPool = new CognitoUserPool("poolID", "clientID", provider);
    CognitoUser user = new CognitoUser("username", "clientID", userPool, provider);
    InitiateSrpAuthRequest authRequest = new InitiateSrpAuthRequest()
    {
        Password = "userPassword"
    };

    AuthFlowResponse authResponse = await
 user.StartWithSrpAuthAsync(authRequest).ConfigureAwait(false);
    accessToken = authResponse.AuthenticationResult.AccessToken;

}
```

Otentikasi dengan tantangan

Melanjutkan alur otentikasi dengan tantangan, seperti dengan NewPasswordRequired dan Multi-Factor Authentication (MFA), juga lebih sederhana. Satu-satunya persyaratan adalah CognitoAuthentication objek, kata sandi pengguna untuk SRP, dan informasi yang diperlukan untuk tantangan berikutnya, yang diperoleh setelah meminta pengguna untuk memasukkannya. Kode berikut menunjukkan satu cara untuk memeriksa jenis tantangan dan mendapatkan respons yang sesuai untuk MFA dan NewPasswordRequired tantangan selama alur otentikasi.

Lakukan permintaan otentikasi dasar seperti sebelumnya, dan`await`. `AuthFlowResponse` Ketika respon diterima loop melalui `AuthenticationResult` objek yang dikembalikan. Jika `ChallengeName` tipenya`NEW_PASSWORD_REQUIRED`, panggil `RespondToNewPasswordRequiredAsync` metodenya.

```
public static async void GetCredsChallengesAsync()
{
    AmazonCognitoIdentityProviderClient provider =
        new AmazonCognitoIdentityProviderClient(new
 Amazon.Runtime.AnonymousAWSCredentials());
    CognitoUserPool userPool = new CognitoUserPool("poolID", "clientID", provider);
    CognitoUser user = new CognitoUser("username", "clientID", userPool, provider);
```

```
    InitiateSrpAuthRequest authRequest = new InitiateSrpAuthRequest(){
        Password = "userPassword"
    };

    AuthFlowResponse authResponse = await
user.StartWithSrpAuthAsync(authRequest).ConfigureAwait(false);

    while (authResponse.AuthenticationResult == null)
    {
        if (authResponse.ChallengeName == ChallengeNameType.NEW_PASSWORD_REQUIRED)
        {
            Console.WriteLine("Enter your desired new password:");
            string newPassword = Console.ReadLine();

            authResponse = await user.RespondToNewPasswordRequiredAsync(new
RespondToNewPasswordRequiredRequest()
            {
                SessionID = authResponse.SessionID,
                NewPassword = newPassword
            });
            accessToken = authResponse.AuthenticationResult.AccessToken;
        }
        else if (authResponse.ChallengeName == ChallengeNameType.SMS_MFA)
        {
            Console.WriteLine("Enter the MFA Code sent to your device:");
            string mfaCode = Console.ReadLine();

            AuthFlowResponse mfaResponse = await user.RespondToSmsMfaAuthAsync(new
RespondToSmsMfaRequest()
            {
                SessionID = authResponse.SessionID,
                MfaCode = mfaCode

            }).ConfigureAwait(false);
            accessToken = authResponse.AuthenticationResult.AccessToken;
        }
        else
        {
            Console.WriteLine("Unrecognized authentication challenge.");
            accessToken = "";
            break;
        }
    }
```

```
    if (authResponse.AuthenticationResult != null)
    {
        Console.WriteLine("User successfully authenticated.");
    }
    else
    {
        Console.WriteLine("Error in authentication process.");
    }

}
```

Gunakan AWS sumber daya setelah otentikasi

Setelah pengguna diautentikasi menggunakan CognitoAuthentication perpustakaan, langkah selanjutnya adalah mengizinkan pengguna mengakses AWS sumber daya yang sesuai. Untuk melakukan ini, Anda harus membuat kumpulan identitas melalui konsol Identitas Federasi Amazon Cognito. Dengan menentukan kumpulan pengguna Amazon Cognito yang Anda buat sebagai penyedia, menggunakan PooLid dan ClientID, Anda dapat mengizinkan pengguna kumpulan pengguna Amazon Cognito mengakses sumber daya yang terhubung ke akun Anda. AWS Anda juga dapat menentukan peran yang berbeda untuk memungkinkan pengguna yang tidak diautentikasi dan yang diautentikasi mengakses sumber daya yang berbeda. Anda dapat mengubah aturan ini di konsol IAM, tempat Anda dapat menambahkan atau menghapus izin di bidang Tindakan kebijakan terlampir peran. Kemudian, dengan menggunakan kumpulan identitas yang sesuai, kumpulan pengguna, dan informasi pengguna Amazon Cognito, Anda dapat melakukan panggilan ke sumber daya yang berbeda AWS . Contoh berikut menunjukkan pengguna yang diautentikasi dengan SRP yang mengakses bucket Amazon S3 berbeda yang diizinkan oleh peran kumpulan identitas terkait

```
public async void GetS3BucketsAsync()
{
    var provider = new AmazonCognitoIdentityProviderClient(new
 AnonymousAWSCredentials());
    CognitoUserPool userPool = new CognitoUserPool("poolID", "clientID", provider);
    CognitoUser user = new CognitoUser("username", "clientID", userPool, provider);

    string password = "userPassword";

    AuthFlowResponse context = await user.StartWithSrpAuthAsync(new
 InitiateSrpAuthRequest()
    {
        Password = password
    }).ConfigureAwait(false);
```

```
    CognitoAWSCredentials credentials =
        user.GetCognitoAWSCredentials("identityPoolID", RegionEndpoint.<
 YourIdentityPoolRegion >);

    using (var client = new AmazonS3Client(credentials))
    {
        ListBucketsResponse response =
            await client.ListBucketsAsync(new
 ListBucketsRequest()).ConfigureAwait(false);

        foreach (S3Bucket bucket in response.Buckets)
        {
            Console.WriteLine(bucket.BucketName);
        }
    }
}
```

Opsi otentikasi lainnya

Selain SRP,, dan MFA NewPasswordRequired, CognitoAuthentication pustaka ekstensi menawarkan aliran otentikasi yang lebih mudah untuk:

- Kustom - Memulai dengan panggilan ke `StartWithCustomAuthAsync(InitiateCustomAuthRequest customRequest)`
- RefreshToken - Memulai dengan panggilan ke `StartWithRefreshTokenAuthAsync(InitiateRefreshTokenAuthRequest refreshTokenRequest)`
- RefreshTokenSRP - Memulai dengan panggilan ke `StartWithRefreshTokenAuthAsync(InitiateRefreshTokenAuthRequest refreshTokenRequest)`
- AdminNoSRP - Memulai dengan panggilan ke `StartWithAdminNoSrpAuthAsync(InitiateAdminNoSrpAuthRequest adminAuthRequest)`

Panggil metode yang sesuai tergantung pada aliran yang Anda inginkan. Kemudian lanjutkan meminta pengguna dengan tantangan seperti yang disajikan dalam `AuthFlowResponse` objek dari setiap panggilan metode. Sebut juga metode respons yang sesuai, seperti

`RespondToSmsMfaAuthAsync` untuk tantangan MFA dan `RespondToCustomAuthAsync` untuk tantangan khusus.

Menggunakan database Amazon DynamoDB NoSQL

> **Note**
>
> Model pemrograman dalam topik ini hadir di .NET Framework dan .NET (Core), tetapi konvensi pemanggilan berbeda, apakah sinkron atau asinkron.

AWS SDK untuk .NET Mendukung Amazon DynamoDB, yang merupakan layanan database NoSQL cepat yang ditawarkan oleh. AWSSDK menyediakan tiga model pemrograman untuk berkomunikasi dengan DynamoDB: model tingkat rendah, model dokumen, dan model persistensi objek.

Informasi berikut memperkenalkan model-model ini dan mereka APIs, memberikan contoh tentang bagaimana dan kapan menggunakannya, dan memberi Anda tautan ke sumber daya pemrograman DynamoDB tambahan di. AWS SDK untuk .NET

Model Tingkat Rendah

Model pemrograman tingkat rendah membungkus panggilan langsung ke layanan DynamoDB. Anda mengakses model ini melalui namespace DBv2Amazon.Dynamo.

Dari ketiga model tersebut, model tingkat rendah mengharuskan Anda untuk menulis kode paling banyak. Misalnya, Anda harus mengonversi tipe data.NET ke padanannya di DynamoDB. Namun, model ini memberi Anda akses ke sebagian besar fitur.

Contoh berikut menunjukkan cara menggunakan model tingkat rendah untuk membuat tabel, memodifikasi tabel, dan menyisipkan item ke dalam tabel di DynamoDB.

Membuat Tabel

Dalam contoh berikut, Anda membuat tabel dengan menggunakan `CreateTable` metode `AmazonDynamoDBClient` kelas. `CreateTable`Metode ini menggunakan instance `CreateTableRequest` kelas yang berisi karakteristik seperti nama atribut item yang diperlukan, definisi kunci primer, dan kapasitas throughput. `CreateTable`Metode mengembalikan sebuah instance dari `CreateTableResponse` kelas.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;
```

```
var client = new AmazonDynamoDBClient();

Console.WriteLine("Getting list of tables");
List<string> currentTables = client.ListTables().TableNames;
Console.WriteLine("Number of tables: " + currentTables.Count);
if (!currentTables.Contains("AnimalsInventory"))
{
    var request = new CreateTableRequest
    {
        TableName = "AnimalsInventory",
        AttributeDefinitions = new List<AttributeDefinition>
      {
        new AttributeDefinition
        {
          AttributeName = "Id",
          // "S" = string, "N" = number, and so on.
          AttributeType = "N"
        },
        new AttributeDefinition
        {
          AttributeName = "Type",
          AttributeType = "S"
        }
      },
        KeySchema = new List<KeySchemaElement>
      {
        new KeySchemaElement
        {
          AttributeName = "Id",
          // "HASH" = hash key, "RANGE" = range key.
          KeyType = "HASH"
        },
        new KeySchemaElement
        {
          AttributeName = "Type",
          KeyType = "RANGE"
        },
      },
        ProvisionedThroughput = new ProvisionedThroughput
        {
            ReadCapacityUnits = 10,
            WriteCapacityUnits = 5
        },
```

```
    };

    var response = client.CreateTable(request);

    Console.WriteLine("Table created with request ID: " +
      response.ResponseMetadata.RequestId);
}
```

Memverifikasi Bahwa Tabel Siap Dimodifikasi

Sebelum Anda dapat mengubah atau memodifikasi tabel, tabel harus siap untuk modifikasi. Contoh berikut menunjukkan cara menggunakan model tingkat rendah untuk memverifikasi bahwa tabel di DynamoDB sudah siap. Dalam contoh ini, tabel target untuk memeriksa direferensikan melalui `DescribeTable` metode `AmazonDynamoDBClient` kelas. Setiap lima detik, kode memeriksa nilai `TableStatus` properti tabel. Ketika status diatur ke`ACTIVE`, tabel siap untuk dimodifikasi.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();
var status = "";

do
{
  // Wait 5 seconds before checking (again).
  System.Threading.Thread.Sleep(TimeSpan.FromSeconds(5));

  try
  {
    var response = client.DescribeTable(new DescribeTableRequest
    {
      TableName = "AnimalsInventory"
    });

    Console.WriteLine("Table = {0}, Status = {1}",
      response.Table.TableName,
      response.Table.TableStatus);

    status = response.Table.TableStatus;
  }
  catch (ResourceNotFoundException)
  {
    // DescribeTable is eventually consistent. So you might
```

```
    //   get resource not found.
  }

} while (status != TableStatus.ACTIVE);
```

Memasukkan Item ke dalam Tabel

Dalam contoh berikut, Anda menggunakan model tingkat rendah untuk menyisipkan dua item ke dalam tabel di DynamoDB. Setiap item dimasukkan melalui `PutItem` metode `AmazonDynamoDBClient` kelas, menggunakan instance `PutItemRequest` kelas. Masing-masing dari dua contoh `PutItemRequest` kelas mengambil nama tabel tempat item akan dimasukkan, dengan serangkaian nilai atribut item.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();

var request1 = new PutItemRequest
{
  TableName = "AnimalsInventory",
  Item = new Dictionary<string, AttributeValue>
  {
    { "Id", new AttributeValue { N = "1" }},
    { "Type", new AttributeValue { S = "Dog" }},
    { "Name", new AttributeValue { S = "Fido" }}
  }
};

var request2 = new PutItemRequest
{
  TableName = "AnimalsInventory",
  Item = new Dictionary<string, AttributeValue>
  {
    { "Id", new AttributeValue { N = "2" }},
    { "Type", new AttributeValue { S = "Cat" }},
    { "Name", new AttributeValue { S = "Patches" }}
  }
};

client.PutItem(request1);
client.PutItem(request2);
```

Model Dokumen

Model pemrograman dokumen menyediakan cara yang lebih mudah untuk bekerja dengan data di DynamoDB. Model ini secara khusus ditujukan untuk mengakses tabel dan item dalam tabel. Anda mengakses model ini melalui DBv2Amazon.Dynamo. DocumentModelnamespace.

Dibandingkan dengan model pemrograman tingkat rendah, model dokumen lebih mudah dikodekan terhadap data DynamoDB. Misalnya, Anda tidak perlu mengonversi sebanyak mungkin tipe data.NET ke padanannya di DynamoDB. Namun, model ini tidak menyediakan akses ke banyak fitur seperti model pemrograman tingkat rendah. Misalnya, Anda dapat menggunakan model ini untuk membuat, mengambil, memperbarui, dan menghapus item dalam tabel. Namun, untuk membuat tabel, Anda harus menggunakan model tingkat rendah. Dibandingkan dengan model persistensi objek, model ini mengharuskan Anda untuk menulis lebih banyak kode untuk menyimpan, memuat, dan menanyakan objek.NET.

Untuk informasi selengkapnya tentang model pemrograman dokumen DynamoDB, lihat .NET: Model dokumen dalam Panduan Pengembang Amazon DynamoDB.

Bagian berikut memberikan informasi tentang cara membuat representasi tabel DynamoDB yang diinginkan, dan contoh tentang cara menggunakan model dokumen untuk menyisipkan item ke dalam tabel dan mendapatkan item dari tabel.

Buat representasi tabel

Untuk melakukan operasi data menggunakan model dokumen, Anda harus terlebih dahulu membuat instance `Table` kelas yang mewakili tabel tertentu. Ada dua cara utama untuk melakukan ini.

LoadTable metode

Mekanisme pertama adalah menggunakan salah satu `LoadTable` metode statis `Table`kelas, mirip dengan contoh berikut:

```
var client = new AmazonDynamoDBClient();
Table table = Table.LoadTable(client, "Reply");
```

> **Note**
>
> Meskipun mekanisme ini berfungsi, dalam kondisi tertentu, kadang-kadang dapat menyebabkan latensi tambahan atau kebuntuan karena perilaku start dingin dan kumpulan

ulir. Untuk informasi lebih lanjut tentang perilaku ini, lihat posting blog Peningkatan Pola Inisialisasi DynamoDB untuk. AWS SDK untuk .NET

TableBuilder

Mekanisme alternatif, `TableBuilder`kelas, diperkenalkan dalam versi 3.7.203 dari paket.Dynamo. AWSSDK DBv2 NuGet Mekanisme ini dapat mengatasi perilaku yang disebutkan di atas dengan menghapus panggilan metode implisit tertentu; khususnya, `DescribeTable` metode. Mekanisme ini digunakan dengan cara yang mirip dengan contoh berikut:

```
var client = new AmazonDynamoDBClient();
var table = new TableBuilder(client, "Reply")
    .AddHashKey("Id", DynamoDBEntryType.String)
    .AddRangeKey("ReplyDateTime", DynamoDBEntryType.String)
    .AddGlobalSecondaryIndex("PostedBy-Message-index", "Author",
 DynamoDBEntryType.String, "Message", DynamoDBEntryType.String)
    .Build();
```

Untuk informasi lebih lanjut tentang mekanisme alternatif ini, lihat lagi posting blog Peningkatan Pola Inisialisasi DynamoDB untuk. AWS SDK untuk .NET

Memasukkan item ke dalam tabel

Dalam contoh berikut, balasan dimasukkan ke dalam tabel Balas melalui `PutItemAsync` metode `Table` kelas. `PutItemAsync`Metode ini mengambil contoh dari `Document` kelas; `Document` kelas hanyalah kumpulan atribut yang diinisialisasi.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DocumentModel;

// Create a representation of the "Reply" table
//  by using one of the mechanisms described previously.

// Then, add a reply to the table.
var newReply = new Document();
newReply["Id"] = Guid.NewGuid().ToString();
newReply["ReplyDateTime"] = DateTime.UtcNow;
newReply["PostedBy"] = "Author1";
newReply["Message"] = "Thank you!";
```

```
await table.PutItemAsync(newReply);
```

Mendapatkan item dari tabel

Dalam contoh berikut, balasan diambil melalui `GetItemAsync` metode `Table` kelas. Untuk menentukan balasan yang akan didapat, `GetItemAsync` metode ini menggunakan kunci hash-and-range utama dari balasan target.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DocumentModel;

// Create a representation of the "Reply" table
//  by using one of the mechanisms described previously.

// Then, get a reply from the table
//  where "guid" is the hash key and "datetime" is the range key.
var reply = await table.GetItemAsync(guid, datetime);
Console.WriteLine("Id = " + reply["Id"]);
Console.WriteLine("ReplyDateTime = " + reply["ReplyDateTime"]);
Console.WriteLine("PostedBy = " + reply["PostedBy"]);
Console.WriteLine("Message = " + reply["Message"]);
```

Contoh sebelumnya secara implisit mengubah nilai tabel menjadi string untuk metode ini. `WriteLine` Anda dapat melakukan konversi eksplisit dengan menggunakan berbagai metode “As [type]” kelas. `DynamoDBEntry` Misalnya, Anda dapat secara eksplisit mengonversi nilai `Id` dari tipe `Primitive` data ke GUID melalui metode: `AsGuid()`

```
var guid = reply["Id"].AsGuid();
```

Model Kegigihan Objek

Model pemrograman persistensi objek dirancang khusus untuk menyimpan, memuat, dan menanyakan objek.NET di DynamoDB. Anda mengakses model ini melalui DBv2Amazon.Dynamo. DataModelnamespace.

Dari ketiga model tersebut, model persistensi objek adalah yang paling mudah untuk dikodekan setiap kali Anda menyimpan, memuat, atau menanyakan data DynamoDB. Misalnya, Anda bekerja dengan tipe data DynamoDB secara langsung. Namun, model ini hanya menyediakan akses ke operasi yang menyimpan, memuat, dan menanyakan objek.NET di DynamoDB. Misalnya, Anda dapat menggunakan model ini untuk membuat, mengambil, memperbarui, dan menghapus item

dalam tabel. Namun, Anda harus terlebih dahulu membuat tabel Anda menggunakan model tingkat rendah, dan kemudian menggunakan model ini untuk memetakan kelas.NET Anda ke tabel.

Untuk informasi selengkapnya tentang model pemrograman persistensi objek DynamoDB, lihat .NET: Model persistensi objek di Panduan Pengembang Amazon DynamoDB.

Contoh berikut menunjukkan kepada Anda bagaimana mendefinisikan kelas.NET yang mewakili item DynamoDB, menggunakan instance dari kelas.NET untuk menyisipkan item ke dalam tabel DynamoDB, dan menggunakan instance dari kelas.NET untuk mendapatkan item dari tabel.

Mendefinisikan kelas.NET yang mewakili item dalam tabel

Dalam contoh berikut dari definisi kelas, `DynamoDBTable` atribut menentukan nama tabel, sedangkan `DynamoDBHashKey` dan `DynamoDBRangeKey` atribut model kunci hash-and-range utama tabel. `DynamoDBGlobalSecondaryIndexHashKey`Atribut didefinisikan sehingga kueri untuk balasan oleh penulis tertentu dapat dibangun.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;

[DynamoDBTable("Reply")]
public class Reply
{
    [DynamoDBHashKey]
    public string Id { get; set; }

    [DynamoDBRangeKey(StoreAsEpoch = false)]
    public DateTime ReplyDateTime { get; set; }

    [DynamoDBGlobalSecondaryIndexHashKey("PostedBy-Message-Index",
        AttributeName ="PostedBy")]
    public string Author { get; set; }

    [DynamoDBGlobalSecondaryIndexRangeKey("PostedBy-Message-Index")]
    public string Message { get; set; }
}
```

Membuat konteks untuk model ketekunan objek

Untuk menggunakan model pemrograman persistensi objek untuk DynamoDB, Anda harus membuat konteks, yang menyediakan koneksi ke DynamoDB dan memungkinkan Anda mengakses tabel, melakukan berbagai operasi, dan menjalankan kueri.

Konteks dasar

Contoh berikut menunjukkan cara membuat konteks yang paling dasar.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;

var client = new AmazonDynamoDBClient();
var context = new DynamoDBContext(client);
```

Konteks dengan DisableFetchingTableMetadata properti

Contoh berikut menunjukkan bagaimana Anda mungkin juga mengatur `DisableFetchingTableMetadata` properti `DynamoDBContextConfig` kelas untuk mencegah panggilan implisit ke `DescribeTable` metode.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;

var client = new AmazonDynamoDBClient();
var context = new DynamoDBContext(client, new DynamoDBContextConfig
{
    DisableFetchingTableMetadata = true
});
```

Jika `DisableFetchingTableMetadata` properti diatur ke `false` (default), seperti yang ditunjukkan pada contoh pertama, Anda dapat menghilangkan atribut yang menggambarkan kunci dan struktur indeks item tabel dari `Reply` kelas. Atribut ini malah akan disimpulkan melalui panggilan implisit ke metode. `DescribeTable` Jika `DisableFetchingTableMetadata` diatur ke`true`, seperti yang ditunjukkan pada contoh kedua, metode model ketekunan objek seperti `SaveAsync` dan `QueryAsync` bergantung sepenuhnya pada atribut yang didefinisikan di `Reply` kelas. Dalam hal ini, panggilan ke `DescribeTable` metode tidak terjadi.

> **Note**
>
> Dalam kondisi tertentu, panggilan ke `DescribeTable` metode terkadang dapat menyebabkan latensi tambahan atau kebuntuan karena perilaku cold-start dan thread-pool. Untuk alasan ini, terkadang menguntungkan untuk menghindari panggilan ke metode itu. Untuk informasi lebih lanjut tentang perilaku ini, lihat posting blog Peningkatan Pola Inisialisasi DynamoDB untuk. AWS SDK untuk .NET

Menggunakan instance dari kelas.NET untuk menyisipkan item ke dalam tabel

Dalam contoh ini, item dimasukkan melalui `SaveAsync` metode `DynamoDBContext` kelas, yang mengambil instance inisialisasi dari kelas.NET yang mewakili item.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;

// Create an appropriate context for the object persistence programming model,
//  examples of which have been described earlier.

// Create an object that represents the new item.
var reply = new Reply()
{
    Id = Guid.NewGuid().ToString(),
    ReplyDateTime = DateTime.UtcNow,
    Author = "Author1",
    Message = "Thank you!"
};

// Insert the item into the table.
await context.SaveAsync<Reply>(reply, new DynamoDBOperationConfig
{
    IndexName = "PostedBy-Message-index"
});
```

Menggunakan instance dari kelas.NET untuk mendapatkan item dari tabel

Dalam contoh ini, kueri dibuat untuk menemukan semua catatan “Author1" dengan menggunakan `QueryAsync` metode `DynamoDBContext` kelas. Kemudian, item diambil melalui `GetNextSetAsync` metode query.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;

// Create an appropriate context for the object persistence programming model,
//  examples of which have been described earlier.

// Construct a query that finds all replies by a specific author.
var query = context.QueryAsync<Reply>("Author1", new DynamoDBOperationConfig
{
    IndexName = "PostedBy-Message-index"
```

```
});

// Display the result.
var set = await query.GetNextSetAsync();
foreach (var item in set)
{
    Console.WriteLine("Id = " + item.Id);
    Console.WriteLine("ReplyDateTime = " + item.ReplyDateTime);
    Console.WriteLine("PostedBy = " + item.Author);
    Console.WriteLine("Message = " + item.Message);
}
```

Informasi tambahan tentang model persistensi objek

Contoh dan penjelasan yang ditunjukkan di atas terkadang menyertakan properti `DynamoDBContext` kelas yang disebut`DisableFetchingTableMetadata`. Properti ini, yang diperkenalkan dalam DBv2 NuGet paket AWSSDK .Dynamo versi 3.7.203, memungkinkan Anda menghindari kondisi tertentu yang dapat menyebabkan latensi tambahan atau kebuntuan karena perilaku cold-start dan thread-pool. Untuk informasi lebih lanjut, lihat posting blog Peningkatan Pola Inisialisasi DynamoDB untuk. AWS SDK untuk .NET

Berikut ini adalah beberapa informasi tambahan tentang properti ini.

- Properti ini dapat diatur secara global dalam `web.config` file Anda `app.config` atau jika Anda menggunakan .NET Framework.
- Properti ini dapat diatur secara global dengan menggunakan `AWSConfigsDynamoDB`kelas, seperti yang ditunjukkan pada contoh berikut.

  ```
  // Set the DisableFetchingTableMetadata property globally
  // before constructing any context objects.
  AWSConfigsDynamoDB.Context.DisableFetchingTableMetadata = true;

  var client = new AmazonDynamoDBClient();
  var context = new DynamoDBContext(client);
  ```

- Dalam beberapa kasus, Anda tidak dapat menambahkan atribut DynamoDB ke kelas.NET; misalnya, jika kelas didefinisikan dalam dependensi. Dalam kasus seperti itu, masih mungkin untuk mengambil keuntungan dari `DisableFetchingTableMetadata` properti. Untuk melakukannya, gunakan `TableBuilder`kelas selain `DisableFetchingTableMetadata` properti. `TableBuilder`Kelas ini juga diperkenalkan dalam versi 3.7.203 dari paket.Dynamo. AWSSDK DBv2 NuGet

```
// Set the DisableFetchingTableMetadata property globally
// before constructing any context objects.
AWSConfigsDynamoDB.Context.DisableFetchingTableMetadata = true;

var client = new AmazonDynamoDBClient();
var context = new DynamoDBContext(client);

var table = new TableBuilder(client, "Reply")
    .AddHashKey("Id", DynamoDBEntryType.String)
    .AddRangeKey("ReplyDateTime", DynamoDBEntryType.String)
    .AddGlobalSecondaryIndex("PostedBy-Message-index", "Author",
 DynamoDBEntryType.String,
        "Message", DynamoDBEntryType.String)
    .Build();

// This registers the "Reply" table we constructed via the builder.
context.RegisterTableDefinition(table);

// Now operations like this will work,
// even if the Reply class was not annotated with this index.
var query = context.QueryAsync<Reply>("Author1", new DynamoDBOperationConfig()
{
    IndexName = "PostedBy-Message-index"
});
```

Informasi selengkapnya

Menggunakan AWS SDK untuk .NET to program DynamoDB, informasi dan contoh

- DynamoDB APIs
- Kickoff Seri DynamoDB
- DynamoDB Series - Model Dokumen
- DynamoDB Series - Skema Konversi
- Seri DynamoDB - Model Persistensi Objek
- Seri DynamoDB - Ekspresi
- Menggunakan Ekspresi dengan Amazon DynamoDB dan AWS SDK untuk .NET
- Dukungan JSON di Amazon DynamoDB

Model, informasi, dan contoh Tingkat Rendah

- Bekerja dengan Tabel Menggunakan API AWS SDK untuk .NET Tingkat Rendah
- Bekerja dengan Item Menggunakan API AWS SDK untuk .NET Tingkat Rendah
- Mengkueri Tabel Menggunakan API Tingkat AWS SDK untuk .NET Rendah
- Memindai Tabel Menggunakan API AWS SDK untuk .NET Tingkat Rendah
- Bekerja dengan Indeks Sekunder Lokal Menggunakan API Tingkat AWS SDK untuk .NET Rendah
- Bekerja dengan Indeks Sekunder Global Menggunakan API Tingkat AWS SDK untuk .NET Rendah

Model dokumen, informasi dan contoh

- Jenis Data DynamoDB
- Dinamo DBEntry
- .NET: Model Dokumen

Model persistensi objek, informasi dan contoh

- .NET: Model Kegigihan Objek

Informasi bermanfaat lainnya

- Lihat Integrasi AWS dengan .NET Aspire untuk informasi tentang pengembangan dengan Amazon DynamoDB lokal melalui .NET Aspire.

Topik

- Menggunakan Ekspresi dengan Amazon DynamoDB dan AWS SDK untuk .NET
- Dukungan JSON di Amazon DynamoDB

Menggunakan Ekspresi dengan Amazon DynamoDB dan AWS SDK untuk .NET

> **Note**
>
> Informasi dalam topik ini khusus untuk proyek berdasarkan .NET Framework dan AWS SDK untuk .NET versi 3.3 dan sebelumnya.

Contoh kode berikut menunjukkan bagaimana menggunakan AWS SDK untuk .NET untuk program DynamoDB dengan ekspresi. Ekspresi menunjukkan atribut yang ingin Anda baca dari item dalam tabel DynamoDB. Anda juga menggunakan ekspresi saat menulis item, untuk menunjukkan kondisi apa pun yang harus dipenuhi (juga dikenal sebagai pembaruan bersyarat) dan bagaimana atribut akan diperbarui. Beberapa contoh pembaruan mengganti atribut dengan nilai baru, atau menambahkan data baru ke daftar atau peta. Untuk informasi selengkapnya, lihat Membaca dan Menulis Item Menggunakan Ekspresi.

Topik

- Sampel Data
- Dapatkan Item Tunggal dengan Menggunakan Ekspresi dan Kunci Utama Item
- Dapatkan Beberapa Item dengan Menggunakan Ekspresi dan Kunci Utama Tabel
- Dapatkan Beberapa Item dengan Menggunakan Ekspresi dan Atribut Item Lainnya
- Cetak Item
- Membuat atau Mengganti Item dengan Menggunakan Ekspresi
- Memperbarui Item dengan Menggunakan Ekspresi
- Menghapus Item dengan Menggunakan Ekspresi
- Info Selengkapnya

Sampel Data

Contoh kode dalam topik ini bergantung pada dua item contoh berikut dalam tabel DynamoDB bernama. `ProductCatalog` Barang-barang ini menjelaskan informasi tentang entri produk dalam katalog toko sepeda fiktif. Item ini didasarkan pada contoh yang diberikan dalam Studi Kasus: ProductCatalog Item. Deskriptor tipe data seperti`B`,`BOOL`,`L`,`M`,`N`, `NS`, dan `SS` sesuai dengan yang ada di Format Data JSON.

```
{
  "Id": {
    "N": "205"
  },
  "Title": {
    "S": "20-Bicycle 205"
  },
  "Description": {
    "S": "205 description"
  },
```

```
  "BicycleType": {
    "S": "Hybrid"
  },
  "Brand": {
    "S": "Brand-Company C"
  },
  "Price": {
    "N": "500"
  },
  "Gender": {
    "S": "B"
  },
  "Color": {
    "SS": [
      "Red",
      "Black"
    ]
  },
  "ProductCategory": {
    "S": "Bike"
  },
  "InStock": {
    "BOOL": true
  },
  "QuantityOnHand": {
    "N": "1"
  },
  "RelatedItems": {
    "NS": [
      "341",
      "472",
      "649"
    ]
  },
  "Pictures": {
    "L": [
      {
        "M": {
          "FrontView": {
            "S": "http://example/products/205_front.jpg"
          }
        }
      },
      {
```

```
        "M": {
          "RearView": {
            "S": "http://example/products/205_rear.jpg"
          }
        }
      },
      {
        "M": {
          "SideView": {
            "S": "http://example/products/205_left_side.jpg"
          }
        }
      }
    ]
  },
  "ProductReviews": {
    "M": {
      "FiveStar": {
        "SS": [
          "Excellent! Can't recommend it highly enough! Buy it!",
          "Do yourself a favor and buy this."
        ]
      },
      "OneStar": {
        "SS": [
          "Terrible product! Do not buy this."
        ]
      }
    }
  }
},
{
  "Id": {
    "N": "301"
  },
  "Title": {
    "S": "18-Bicycle 301"
  },
  "Description": {
    "S": "301 description"
  },
  "BicycleType": {
    "S": "Road"
  },
```

```
  "Brand": {
    "S": "Brand-Company C"
  },
  "Price": {
    "N": "185"
  },
  "Gender": {
    "S": "F"
  },
  "Color": {
    "SS": [
      "Blue",
      "Silver"
    ]
  },
  "ProductCategory": {
    "S": "Bike"
  },
  "InStock": {
    "BOOL": true
  },
  "QuantityOnHand": {
    "N": "3"
  },
  "RelatedItems": {
    "NS": [
      "801",
      "822",
      "979"
    ]
  },
  "Pictures": {
    "L": [
      {
        "M": {
          "FrontView": {
            "S": "http://example/products/301_front.jpg"
          }
        }
      },
      {
        "M": {
          "RearView": {
            "S": "http://example/products/301_rear.jpg"
```

```
          }
        }
      },
      {
        "M": {
          "SideView": {
            "S": "http://example/products/301_left_side.jpg"
          }
        }
      }
    ]
  },
  "ProductReviews": {
    "M": {
      "FiveStar": {
        "SS": [
          "My daughter really enjoyed this bike!"
        ]
      },
      "ThreeStar": {
        "SS": [
          "This bike was okay, but I would have preferred it in my color.",
       "Fun to ride."
        ]
      }
    }
  }
}
```

Dapatkan Item Tunggal dengan Menggunakan Ekspresi dan Kunci Utama Item

Contoh berikut menampilkan `Amazon.DynamoDBv2.AmazonDynamoDBClient.GetItem` metode dan satu set ekspresi untuk mendapatkan dan kemudian mencetak item yang memiliki `Id` dari`205`. Hanya atribut berikut dari item yang dikembalikan:`Id`,`Title`,`Description`, `ColorRelatedItems`,`Pictures`, dan`ProductReviews`.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();
var request = new GetItemRequest
{
  TableName = "ProductCatalog",
```

```
  ProjectionExpression = "Id, Title, Description, Color, #ri, Pictures, #pr",
  ExpressionAttributeNames = new Dictionary<string, string>
  {
    { "#pr", "ProductReviews" },
    { "#ri", "RelatedItems" }
  },
  Key = new Dictionary<string, AttributeValue>
  {
    { "Id", new AttributeValue { N = "205" } }
  },
};
var response = client.GetItem(request);

// PrintItem() is a custom function.
PrintItem(response.Item);
```

Dalam contoh sebelumnya, `ProjectionExpression` properti menentukan atribut yang akan dikembalikan. `ExpressionAttributeNames`Properti menentukan placeholder `#pr` untuk mewakili `ProductReviews` atribut dan placeholder `#ri` untuk mewakili atribut. `RelatedItems` Panggilan untuk `PrintItem` merujuk ke fungsi kustom seperti yang dijelaskan dalam Cetak Item.

Dapatkan Beberapa Item dengan Menggunakan Ekspresi dan Kunci Utama Tabel

Contoh berikut menampilkan `Amazon.DynamoDBv2.AmazonDynamoDBClient.Query` metode dan satu set ekspresi untuk mendapatkan dan kemudian mencetak item yang memiliki `Id` dari`301`, tetapi hanya jika nilai `Price` lebih besar dari`150`. Hanya atribut berikut dari item yang dikembalikan: `IdTitle`,, dan semua `ThreeStar` atribut di`ProductReviews`.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();
var request = new QueryRequest
{
  TableName = "ProductCatalog",
  KeyConditions = new Dictionary<string,Condition>
  {
    { "Id", new Condition()
      {
        ComparisonOperator = ComparisonOperator.EQ,
        AttributeValueList = new List<AttributeValue>
        {
```

```
          new AttributeValue { N = "301" }
        }
      }
    }
  },
  ProjectionExpression = "Id, Title, #pr.ThreeStar",
  ExpressionAttributeNames = new Dictionary<string, string>
  {
    { "#pr", "ProductReviews" },
    { "#p", "Price" }
  },
  ExpressionAttributeValues = new Dictionary<string,AttributeValue>
  {
    { ":val", new AttributeValue { N = "150" } }
  },
  FilterExpression = "#p > :val"
};
var response = client.Query(request);

foreach (var item in response.Items)
{
  // Write out the first page of an item's attribute keys and values.
  // PrintItem() is a custom function.
  PrintItem(item);
  Console.WriteLine("=====");
}
```

Dalam contoh sebelumnya, `ProjectionExpression` properti menentukan atribut yang akan dikembalikan. `ExpressionAttributeNames`Properti menentukan placeholder `#pr` untuk mewakili `ProductReviews` atribut dan placeholder #p untuk mewakili atribut. `Price` `#pr.ThreeStar`menentukan untuk mengembalikan hanya `ThreeStar` atribut. `ExpressionAttributeValues`Properti menentukan placeholder `:val` untuk mewakili nilai. `150` `FilterExpression`Properti menentukan bahwa #p (`Price`) harus lebih besar dari `:val` (`150`). Panggilan untuk `PrintItem` merujuk ke fungsi kustom seperti yang dijelaskan dalam Cetak Item.

Dapatkan Beberapa Item dengan Menggunakan Ekspresi dan Atribut Item Lainnya

Contoh berikut menampilkan `Amazon.DynamoDBv2.AmazonDynamoDBClient.Scan` metode dan satu set ekspresi untuk mendapatkan dan kemudian mencetak semua item yang memiliki `ProductCategory` dari`Bike`. Hanya atribut berikut dari item yang dikembalikan: `IdTitle`,, dan semua atribut di`ProductReviews`.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();
var request = new ScanRequest
{
  TableName = "ProductCatalog",
  ProjectionExpression = "Id, Title, #pr",
  ExpressionAttributeValues = new Dictionary<string,AttributeValue>
  {
    { ":catg", new AttributeValue { S = "Bike" } }
  },
  ExpressionAttributeNames = new Dictionary<string, string>
  {
    { "#pr", "ProductReviews" },
    { "#pc", "ProductCategory" }
  },
  FilterExpression = "#pc = :catg",
};
var response = client.Scan(request);

foreach (var item in response.Items)
{
  // Write out the first page/scan of an item's attribute keys and values.
  // PrintItem() is a custom function.
  PrintItem(item);
  Console.WriteLine("=====");
}
```

Dalam contoh sebelumnya, `ProjectionExpression` properti menentukan atribut yang akan dikembalikan. `ExpressionAttributeNames`Properti menentukan placeholder `#pr` untuk mewakili `ProductReviews` atribut dan placeholder `#pc` untuk mewakili atribut. `ProductCategory` `ExpressionAttributeValues`Properti menentukan placeholder `:catg` untuk mewakili nilai. `Bike` `FilterExpression`Properti menentukan bahwa `#pc` (`ProductCategory`) harus sama dengan `:catg` (`Bike`). Panggilan untuk `PrintItem` merujuk ke fungsi kustom seperti yang dijelaskan dalam Cetak Item.

Cetak Item

Contoh berikut menunjukkan bagaimana untuk mencetak atribut item dan nilai-nilai. Contoh ini digunakan dalam contoh sebelumnya yang menunjukkan cara Mendapatkan Item Tunggal dengan Menggunakan Ekspresi dan Kunci Utama Item, Dapatkan Beberapa Item dengan Menggunakan

Ekspresi dan Kunci Utama Tabel, dan Dapatkan Beberapa Item dengan Menggunakan Ekspresi dan Atribut Item Lainnya.

```
// using Amazon.DynamoDBv2.Model;

// Writes out an item's attribute keys and values.
public static void PrintItem(Dictionary<string, AttributeValue> attrs)
{
  foreach (KeyValuePair<string, AttributeValue> kvp in attrs)
  {
    Console.Write(kvp.Key + " = ");
    PrintValue(kvp.Value);
  }
}

// Writes out just an attribute's value.
public static void PrintValue(AttributeValue value)
{
  // Binary attribute value.
  if (value.B != null)
  {
    Console.Write("Binary data");
  }
  // Binary set attribute value.
  else if (value.BS.Count > 0)
  {
    foreach (var bValue in value.BS)
    {
      Console.Write("\n  Binary data");
    }
  }
  // List attribute value.
  else if (value.L.Count > 0)
  {
    foreach (AttributeValue attr in value.L)
    {
      PrintValue(attr);
    }
  }
  // Map attribute value.
  else if (value.M.Count > 0)
  {
    Console.Write("\n");
```

```
    PrintItem(value.M);
  }
  // Number attribute value.
  else if (value.N != null)
  {
    Console.Write(value.N);
  }
  // Number set attribute value.
  else if (value.NS.Count > 0)
  {
    Console.Write("{0}", string.Join("\n", value.NS.ToArray()));
  }
  // Null attribute value.
  else if (value.NULL)
  {
    Console.Write("Null");
  }
  // String attribute value.
  else if (value.S != null)
  {
    Console.Write(value.S);
  }
  // String set attribute value.
  else if (value.SS.Count > 0)
  {
    Console.Write("{0}", string.Join("\n", value.SS.ToArray()));
  }
  // Otherwise, boolean value.
  else
  {
    Console.Write(value.BOOL);
  }

  Console.Write("\n");
}
```

Dalam contoh sebelumnya, setiap nilai atribut memiliki beberapa data-type-specific properti yang dapat dievaluasi untuk menentukan format yang benar untuk mencetak atribut. Properti ini termasuk`B`,`BOOL`,`BS`,`L`,`M`,`N`,`NS`,`NULL`,`S`, dan`SS`, yang sesuai dengan yang ada dalam Format Data JSON. Untuk properti seperti`B`,`N`,`NULL`, dan`S`, jika properti yang sesuai tidak`null`, maka atribut adalah tipe `null` non-data yang sesuai. Untuk properti seperti`BS`,, `LM`,`NS`, dan`SS`, jika `Count` lebih besar dari nol, maka atributnya adalah tipe non-zero-value data yang sesuai. Jika semua data-type-

specific properti atribut adalah salah satu `null` atau `Count` sama dengan nol, maka atribut sesuai dengan tipe `BOOL` data.

Membuat atau Mengganti Item dengan Menggunakan Ekspresi

Contoh berikut menampilkan `Amazon.DynamoDBv2.AmazonDynamoDBClient.PutItem` metode dan satu set ekspresi untuk memperbarui item yang memiliki a `Title` of`18-Bicycle 301`. Jika item belum ada, item baru ditambahkan.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();
var request = new PutItemRequest
{
  TableName = "ProductCatalog",
  ExpressionAttributeNames = new Dictionary<string, string>
  {
    { "#title", "Title" }
  },
  ExpressionAttributeValues = new Dictionary<string, AttributeValue>
  {
    { ":product", new AttributeValue { S = "18-Bicycle 301" } }
  },
  ConditionExpression = "#title = :product",
  // CreateItemData() is a custom function.
  Item = CreateItemData()
};
client.PutItem(request);
```

Dalam contoh sebelumnya, `ExpressionAttributeNames` properti menentukan placeholder `#title` untuk mewakili atribut. `Title` `ExpressionAttributeValues`Properti menentukan placeholder `:product` untuk mewakili nilai. `18-Bicycle 301` `ConditionExpression`Properti menentukan bahwa `#title` (`Title`) harus sama dengan `:product` (`18-Bicycle 301`). Panggilan untuk `CreateItemData` mengacu pada fungsi kustom berikut:

```
// using Amazon.DynamoDBv2.Model;

// Provides a sample item that can be added to a table.
public static Dictionary<string, AttributeValue> CreateItemData()
{
```

```
  var itemData = new Dictionary<string, AttributeValue>
  {
    { "Id", new AttributeValue { N = "301" } },
    { "Title", new AttributeValue { S = "18\" Girl's Bike" } },
    { "BicycleType", new AttributeValue { S = "Road" } },
    { "Brand" , new AttributeValue { S = "Brand-Company C" } },
    { "Color", new AttributeValue { SS = new List<string>{ "Blue", "Silver" } } },
    { "Description", new AttributeValue { S = "301 description" } },
    { "Gender", new AttributeValue { S = "F" } },
    { "InStock", new AttributeValue { BOOL = true } },
    { "Pictures", new AttributeValue { L = new List<AttributeValue>{
      { new AttributeValue { M = new Dictionary<string,AttributeValue>{
        { "FrontView", new AttributeValue { S = "http://example/
products/301_front.jpg" } } } } },
      { new AttributeValue { M = new Dictionary<string,AttributeValue>{
        { "RearView", new AttributeValue {S = "http://example/
products/301_rear.jpg" } } } } },
      { new AttributeValue { M = new Dictionary<string,AttributeValue>{
        { "SideView", new AttributeValue { S = "http://example/
products/301_left_side.jpg" } } } } }
    } } },
    { "Price", new AttributeValue { N = "185" } },
    { "ProductCategory", new AttributeValue { S = "Bike" } },
    { "ProductReviews", new AttributeValue { M = new Dictionary<string,AttributeValue>{
      { "FiveStar", new AttributeValue { SS = new List<string>{
        "My daughter really enjoyed this bike!" } } },
      { "OneStar", new AttributeValue { SS = new List<string>{
        "Fun to ride.",
        "This bike was okay, but I would have preferred it in my color." } } }
    } } },
    { "QuantityOnHand", new AttributeValue { N = "3" } },
    { "RelatedItems", new AttributeValue { NS = new List<string>{ "979", "822",
 "801" } } }
  };

  return itemData;
}
```

Dalam contoh sebelumnya, item contoh dengan data sampel dikembalikan ke pemanggil. Serangkaian atribut dan nilai yang sesuai dibangun, menggunakan tipe data sepertiBOOL,,L,M,N,NS, dan SSS, yang sesuai dengan yang ada dalam Format Data JSON.

Memperbarui Item dengan Menggunakan Ekspresi

Contoh berikut menampilkan `Amazon.DynamoDBv2.AmazonDynamoDBClient.UpdateItem` metode dan satu set ekspresi untuk mengubah `Title` ke `18" Girl's Bike` untuk item dengan `Id` dari`301`.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();
var request = new UpdateItemRequest
{
  TableName = "ProductCatalog",
  Key = new Dictionary<string,AttributeValue>
  {
     { "Id", new AttributeValue { N = "301" } }
  },
  ExpressionAttributeNames = new Dictionary<string, string>
  {
    { "#title", "Title" }
  },
  ExpressionAttributeValues = new Dictionary<string, AttributeValue>
  {
    { ":newproduct", new AttributeValue { S = "18\" Girl's Bike" } }
  },
  UpdateExpression = "SET #title = :newproduct"
};
client.UpdateItem(request);
```

Dalam contoh sebelumnya, `ExpressionAttributeNames` properti menentukan placeholder `#title` untuk mewakili atribut. `Title` `ExpressionAttributeValues`Properti menentukan placeholder `:newproduct` untuk mewakili nilai. `18" Girl's Bike` `UpdateExpression`Properti menentukan untuk mengubah `#title` (`Title`) ke `:newproduct` (`18" Girl's Bike`).

Menghapus Item dengan Menggunakan Ekspresi

Contoh berikut menampilkan `Amazon.DynamoDBv2.AmazonDynamoDBClient.DeleteItem` metode dan satu set ekspresi untuk menghapus item dengan `Id` dari`301`, tetapi hanya jika item `Title` tersebut`18-Bicycle 301`.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;
```

```
var client = new AmazonDynamoDBClient();
var request = new DeleteItemRequest
{
  TableName = "ProductCatalog",
  Key = new Dictionary<string,AttributeValue>
  {
     { "Id", new AttributeValue { N = "301" } }
  },
  ExpressionAttributeNames = new Dictionary<string, string>
  {
    { "#title", "Title" }
  },
  ExpressionAttributeValues = new Dictionary<string, AttributeValue>
  {
    { ":product", new AttributeValue { S = "18-Bicycle 301" } }
  },
  ConditionExpression = "#title = :product"
};
client.DeleteItem(request);
```

Dalam contoh sebelumnya, `ExpressionAttributeNames` properti menentukan placeholder `#title` untuk mewakili atribut. `Title` `ExpressionAttributeValues`Properti menentukan placeholder `:product` untuk mewakili nilai. `18-Bicycle 301` `ConditionExpression`Properti menentukan bahwa `#title` (`Title`) harus sama `:product` (`18-Bicycle 301`).

Info Selengkapnya

Untuk informasi selengkapnya dan contoh kode, lihat:

- Seri DynamoDB - Ekspresi
- Mengakses Atribut Item dengan Ekspresi Proyeksi
- Menggunakan Placeholder untuk Nama dan Nilai Atribut
- Menentukan Kondisi dengan Ekspresi Kondisi
- Memodifikasi Item dan Atribut dengan Ekspresi Pembaruan
- Bekerja dengan Item Menggunakan API AWS SDK untuk .NET Tingkat Rendah
- Mengkueri Tabel Menggunakan API Tingkat AWS SDK untuk .NET Rendah
- Memindai Tabel Menggunakan API AWS SDK untuk .NET Tingkat Rendah
- Bekerja dengan Indeks Sekunder Lokal Menggunakan API Tingkat AWS SDK untuk .NET Rendah

- Bekerja dengan Indeks Sekunder Global Menggunakan API Tingkat AWS SDK untuk .NET Rendah

Dukungan JSON di Amazon DynamoDB

> **Note**
>
> Informasi dalam topik ini khusus untuk proyek berdasarkan .NET Framework dan AWS SDK untuk .NET versi 3.3 dan sebelumnya.

AWS SDK untuk .NET Mendukung data JSON saat bekerja dengan Amazon DynamoDB. Ini memungkinkan Anda untuk lebih mudah mendapatkan data berformat JSON dari, dan memasukkan dokumen JSON ke dalam, tabel DynamoDB.

Topik

- Dapatkan Data dari Tabel DynamoDB dalam Format JSON
- Masukkan Data Format JSON ke dalam Tabel DynamoDB
- DynamoDB Jenis Data Konversi ke JSON
- Info Selengkapnya

Dapatkan Data dari Tabel DynamoDB dalam Format JSON

Contoh berikut menunjukkan bagaimana untuk mendapatkan data dari tabel DynamoDB dalam format JSON:

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.DocumentModel;

var client = new AmazonDynamoDBClient();
var table = Table.LoadTable(client, "AnimalsInventory");
var item = table.GetItem(3, "Horse");

var jsonText = item.ToJson();
Console.Write(jsonText);

// Output:
//   {"Name":"Shadow","Type":"Horse","Id":3}

var jsonPrettyText = item.ToJsonPretty();
```

```
Console.WriteLine(jsonPrettyText);

// Output:
//   {
//     "Name" : "Shadow",
//     "Type" : "Horse",
//     "Id"   : 3
//   }
```

Dalam contoh sebelumnya, `ToJson` metode `Document` kelas mengubah item dari tabel menjadi string berformat JSON. Item diambil melalui `GetItem` metode `Table` kelas. Untuk menentukan item yang akan didapatkan, dalam contoh ini, `GetItem` metode menggunakan kunci hash-and-range utama dari item target. Untuk menentukan tabel untuk mendapatkan item dari, `LoadTable` metode `Table` kelas menggunakan instance dari `AmazonDynamoDBClient` kelas dan nama tabel target di DynamoDB.

Masukkan Data Format JSON ke dalam Tabel DynamoDB

Contoh berikut menunjukkan bagaimana menggunakan format JSON untuk menyisipkan item ke dalam tabel DynamoDB:

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.DocumentModel;

var client = new AmazonDynamoDBClient();
var table = Table.LoadTable(client, "AnimalsInventory");
var jsonText = "{\"Id\":6,\"Type\":\"Bird\",\"Name\":\"Tweety\"}";
var item = Document.FromJson(jsonText);

table.PutItem(item);
```

Dalam contoh sebelumnya, `FromJson` metode `Document` kelas mengkonversi string JSON-diformat menjadi item. Item dimasukkan ke dalam tabel melalui `PutItem` metode `Table` kelas, yang menggunakan instance `Document` kelas yang berisi item. Untuk menentukan tabel untuk memasukkan item ke dalam, `LoadTable` metode `Table` kelas dipanggil, menentukan instance `AmazonDynamoDBClient` kelas dan nama tabel target di DynamoDB.

DynamoDB Jenis Data Konversi ke JSON

Setiap kali Anda memanggil `ToJson` metode `Document` kelas, dan kemudian pada data JSON yang dihasilkan Anda memanggil `FromJson` metode untuk mengubah data JSON kembali ke instance

`Document` kelas, beberapa tipe data DynamoDB tidak akan dikonversi seperti yang diharapkan. Secara khusus:

- DynamoDB set (`SS`NS,, `BS` dan jenis) akan dikonversi ke array JSON.
- Skalar dan set biner DynamoDB (`BB`Sdan tipe) akan dikonversi ke string JSON yang dikodekan base64 atau daftar string.

  Dalam skenario ini, Anda harus memanggil `DecodeBase64Attributes` metode `Document` kelas untuk mengganti data JSON yang dikodekan base64 dengan representasi biner yang benar. Contoh berikut menggantikan atribut item skalar biner berkode base64 dalam contoh `Document` kelas, bernama`Picture`, dengan representasi biner yang benar. Contoh ini juga melakukan hal yang sama untuk atribut item set biner yang dikodekan base64 dalam contoh kelas yang sama, bernama: `Document` `RelatedPictures`

```
item.DecodeBase64Attributes("Picture", "RelatedPictures");
```

Info Selengkapnya

Untuk informasi lebih lanjut dan contoh pemrograman JSON dengan DynamoDB dengan, lihat: AWS SDK untuk .NET

- Dukungan DynamoDB JSON
- Pembaruan Amazon DynamoDB - JSON, Tingkat Gratis yang Diperluas, Penskalaan Fleksibel, Item Lebih Besar

Bekerja dengan Amazon EC2

Ini AWS SDK untuk .NET mendukung Amazon EC2, yang merupakan layanan web yang menyediakan kapasitas komputasi yang dapat diubah ukurannya. Anda menggunakan kapasitas komputasi ini untuk membangun dan meng-host sistem perangkat lunak Anda.

APIs

AWS SDK untuk .NET Ini menyediakan API untuk klien Amazon EC2. API memungkinkan Anda untuk bekerja dengan fitur EC2 seperti grup keamanan dan pasangan kunci. API juga memungkinkan Anda mengontrol instans Amazon EC2. Bagian ini berisi sejumlah kecil contoh yang menunjukkan

pola yang dapat Anda ikuti saat bekerja dengan ini APIs. Untuk melihat set lengkap APIs, lihat Referensi AWS SDK untuk .NET API (dan gulir ke “Amazon.EC2").

Amazon EC2 APIs disediakan oleh AWSSDK NuGet paket.EC2.

Prasyarat

Sebelum Anda mulai, pastikan Anda telah mengatur lingkungan Anda dan mengkonfigurasi proyek Anda. Juga tinjau informasi diMenggunakan SDK .

Tentang contoh

Contoh di bagian ini menunjukkan cara bekerja dengan klien Amazon EC2 dan mengelola instans Amazon EC2.

Tutorial Instans Spot EC2 menunjukkan cara meminta Instans Spot Amazon EC2. Instans Spot memungkinkan Anda mengakses kapasitas EC2 yang tidak digunakan dengan harga kurang dari harga Sesuai Permintaan.

Topik

- Bekerja dengan grup keamanan di Amazon EC2
- Bekerja dengan pasangan kunci Amazon EC2
- Melihat Wilayah Amazon EC2 dan Availability Zone
- Bekerja dengan instans Amazon EC2
- Tutorial Instans Spot Amazon EC2

Bekerja dengan grup keamanan di Amazon EC2

Di Amazon EC2, grup keamanan bertindak sebagai firewall virtual yang mengontrol lalu lintas jaringan untuk satu atau lebih instans EC2. Secara default, EC2 mengaitkan instans Anda dengan grup keamanan yang tidak mengizinkan lalu lintas masuk. Anda dapat membuat grup keamanan yang memungkinkan instans EC2 Anda menerima lalu lintas tertentu. Misalnya, jika Anda perlu terhubung ke instans EC2 Windows, Anda harus mengkonfigurasi grup keamanan untuk memungkinkan lalu lintas RDP.

Untuk membaca selengkapnya tentang grup keamanan, lihat grup keamanan Amazon EC2 di Panduan Pengguna Amazon EC2.

> **Warning**
>
> EC2-Classic sudah tidak digunakan lagi pada 15 Agustus 2022. Kami menyarankan Anda bermigrasi dari EC2-Classic ke VPC. Untuk informasi lebih lanjut, lihat posting blog EC2-Classic Networking is Retiring - Inilah Cara Mempersiapkan.

Untuk informasi tentang APIs dan prasyarat, lihat bagian induk (). Bekerja dengan Amazon EC2

Topik

- Menghitung kelompok keamanan
- Membuat grup keamanan
- Memperbarui grup keamanan

Menghitung kelompok keamanan

Contoh ini menunjukkan kepada Anda cara menggunakan AWS SDK untuk .NET untuk menghitung grup keamanan. Jika Anda menyediakan ID Amazon Virtual Private Cloud, aplikasi menghitung grup keamanan untuk VPC tertentu. Jika tidak, aplikasi hanya menampilkan daftar semua grup keamanan yang tersedia.

Bagian berikut menyediakan cuplikan dari contoh ini. Kode lengkap untuk contoh ditampilkan setelah itu, dan dapat dibangun dan dijalankan apa adanya.

Topik

- Menghitung kelompok keamanan
- Kode lengkap
- Pertimbangan tambahan

Menghitung kelompok keamanan

Cuplikan berikut menyebutkan grup keamanan Anda. Ini menghitung semua grup atau grup untuk VPC tertentu jika diberikan.

Contoh di akhir topik ini menunjukkan cuplikan ini digunakan.

```
    //
    // Method to enumerate the security groups
```

```
    private static async Task EnumerateGroups(IAmazonEC2 ec2Client, string vpcID)
    {
      // A request object, in case we need it.
      var request = new DescribeSecurityGroupsRequest();

      // Put together the properties, if needed
      if(!string.IsNullOrEmpty(vpcID))
      {
        // We have a VPC ID. Find the security groups for just that VPC.
        Console.WriteLine($"\nGetting security groups for VPC {vpcID}...\n");
        request.Filters.Add(new Filter
        {
          Name = "vpc-id",
          Values = new List<string>() { vpcID }
        });
      }

      // Get the list of security groups
      DescribeSecurityGroupsResponse response =
        await ec2Client.DescribeSecurityGroupsAsync(request);

      // Display the list of security groups.
      foreach (SecurityGroup item in response.SecurityGroups)
      {
        Console.WriteLine("Security group: " + item.GroupId);
        Console.WriteLine("\tGroupId: " + item.GroupId);
        Console.WriteLine("\tGroupName: " + item.GroupName);
        Console.WriteLine("\tVpcId: " + item.VpcId);
        Console.WriteLine();
      }
    }
```

Kode lengkap

Bagian ini menunjukkan referensi yang relevan dan kode lengkap untuk contoh ini.

Referensi SDK

NuGet paket:

- AWSSDK.EC2

Elemen pemrograman:

- Namespace Amazon.EC2

  EC2Klien Kelas Amazon

- Namespace Amazon.EC2.Model

  Kelas DescribeSecurityGroupsRequest

  Kelas DescribeSecurityGroupsResponse

  Filter Kelas

  Kelas SecurityGroup

Kode

```
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2EnumerateSecGroups
{
  class Program
  {
    static async Task Main(string[] args)
    {
      // Parse the command line
       string vpcID = string.Empty;
      if(args.Length == 0)
      {
        Console.WriteLine("\nEC2EnumerateSecGroups [vpc_id]");
        Console.WriteLine("  vpc_id - The ID of the VPC for which you want to see
 security groups.");
        Console.WriteLine("\nSince you specified no arguments, showing all available
 security groups.");
      }
      else
      {
        vpcID = args[0];
      }
```

```
      if(vpcID.StartsWith("vpc-") || string.IsNullOrEmpty(vpcID))
      {
        // Create an EC2 client object
        var ec2Client = new AmazonEC2Client();

        // Enumerate the security groups
        await EnumerateGroups(ec2Client, vpcID);
      }
      else
      {
        Console.WriteLine("Could not find a valid VPC ID in the command-line
arguments:");
        Console.WriteLine($"{args[0]}");
      }
    }


    //
    // Method to enumerate the security groups
    private static async Task EnumerateGroups(IAmazonEC2 ec2Client, string vpcID)
    {
      // A request object, in case we need it.
      var request = new DescribeSecurityGroupsRequest();

      // Put together the properties, if needed
      if(!string.IsNullOrEmpty(vpcID))
      {
        // We have a VPC ID. Find the security groups for just that VPC.
        Console.WriteLine($"\nGetting security groups for VPC {vpcID}...\n");
        request.Filters.Add(new Filter
        {
          Name = "vpc-id",
          Values = new List<string>() { vpcID }
        });
      }

      // Get the list of security groups
      DescribeSecurityGroupsResponse response =
        await ec2Client.DescribeSecurityGroupsAsync(request);

      // Display the list of security groups.
      foreach (SecurityGroup item in response.SecurityGroups)
      {
        Console.WriteLine("Security group: " + item.GroupId);
```

```
        Console.WriteLine("\tGroupId: " + item.GroupId);
        Console.WriteLine("\tGroupName: " + item.GroupName);
        Console.WriteLine("\tVpcId: " + item.VpcId);
        Console.WriteLine();
      }
    }
  }
}
```

Pertimbangan tambahan

- Perhatikan untuk kasus VPC bahwa filter dibuat dengan `Name` bagian dari pasangan nama-nilai yang disetel ke “vpc-id”. Nama ini berasal dari deskripsi untuk `Filters` properti DescribeSecurityGroupsRequestkelas.

- Untuk mendapatkan daftar lengkap grup keamanan Anda, Anda juga dapat menggunakan DescribeSecurityGroupsAsync tanpa parameter.

- Anda dapat memverifikasi hasilnya dengan memeriksa daftar grup keamanan di konsol Amazon EC2.

Membuat grup keamanan

Contoh ini menunjukkan cara menggunakan AWS SDK untuk .NET untuk membuat grup keamanan. Anda dapat memberikan ID VPC yang ada untuk membuat grup keamanan EC2 di VPC. Jika Anda tidak memberikan ID seperti itu, grup keamanan baru akan digunakan untuk EC2-Classic jika AWS akun Anda mendukung ini.

Jika Anda tidak memberikan ID VPC dan AWS akun Anda tidak mendukung EC2-Classic, grup keamanan baru akan menjadi milik VPC default akun Anda.

> **Warning**
>
> EC2-Classic sudah tidak digunakan lagi pada 15 Agustus 2022. Kami menyarankan Anda bermigrasi dari EC2-Classic ke VPC. Untuk informasi lebih lanjut, lihat posting blog EC2-Classic Networking is Retiring - Inilah Cara Mempersiapkan.

Bagian berikut menyediakan cuplikan dari contoh ini. Kode lengkap untuk contoh ditampilkan setelah itu, dan dapat dibangun dan dijalankan apa adanya.

Topik

- Temukan grup keamanan yang ada
- Membuat grup keamanan
- Kode lengkap

Temukan grup keamanan yang ada

Cuplikan berikut mencari grup keamanan yang ada dengan nama yang diberikan di VPC yang diberikan.

Contoh di akhir topik ini menunjukkan cuplikan ini digunakan.

```
//
// Method to determine if a security group with the specified name
// already exists in the VPC
private static async Task<List<SecurityGroup>> FindSecurityGroups(
  IAmazonEC2 ec2Client, string groupName, string vpcID)
{
  var request = new DescribeSecurityGroupsRequest();
  request.Filters.Add(new Filter{
    Name = "group-name",
    Values = new List<string>() { groupName }
  });
  if(!string.IsNullOrEmpty(vpcID))
    request.Filters.Add(new Filter{
      Name = "vpc-id",
      Values = new List<string>() { vpcID }
    });

  var response = await ec2Client.DescribeSecurityGroupsAsync(request);
  return response.SecurityGroups;
}
```

Membuat grup keamanan

Cuplikan berikut membuat grup keamanan baru jika grup dengan nama itu tidak ada di VPC yang diberikan. Jika tidak ada VPC yang diberikan dan satu atau lebih grup dengan nama itu ada, cuplikan hanya mengembalikan daftar grup.

Contoh di akhir topik ini menunjukkan cuplikan ini digunakan.

```
    //
    // Method to create a new security group (either EC2-Classic or EC2-VPC)
    // If vpcID is empty, the security group will be for EC2-Classic
    private static async Task<List<SecurityGroup>> CreateSecurityGroup(
      IAmazonEC2 ec2Client, string groupName, string vpcID)
    {
      // See if one or more security groups with that name
      // already exist in the given VPC. If so, return the list of them.
      var securityGroups = await FindSecurityGroups(ec2Client, groupName, vpcID);
      if (securityGroups.Count > 0)
      {
        Console.WriteLine(
          $"\nOne or more security groups with name {groupName} already exist.\n");
        return securityGroups;
      }

      // If the security group doesn't already exists, create it.
      var createRequest = new CreateSecurityGroupRequest{
        GroupName = groupName
      };
      if(string.IsNullOrEmpty(vpcID))
      {
        createRequest.Description = "My .NET example security group for EC2-Classic";
      }
      else
      {
        createRequest.VpcId = vpcID;
        createRequest.Description = "My .NET example security group for EC2-VPC";
      }
      CreateSecurityGroupResponse createResponse =
        await ec2Client.CreateSecurityGroupAsync(createRequest);

      // Return the new security group
      DescribeSecurityGroupsResponse describeResponse =
        await ec2Client.DescribeSecurityGroupsAsync(new DescribeSecurityGroupsRequest{
          GroupIds = new List<string>() { createResponse.GroupId }
        });
      return describeResponse.SecurityGroups;
    }
```

Kode lengkap

Bagian ini menunjukkan referensi yang relevan dan kode lengkap untuk contoh ini.

Referensi SDK

NuGet paket:

- AWSSDK.EC2

Elemen pemrograman:

- Namespace Amazon.EC2

  EC2Klien Kelas Amazon
- Namespace Amazon.EC2.Model

  Kelas CreateSecurityGroupRequest

  Kelas CreateSecurityGroupResponse

  Kelas DescribeSecurityGroupsRequest

  Kelas DescribeSecurityGroupsResponse

  Filter Kelas

  Kelas SecurityGroup

Kodenya

```
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2CreateSecGroup
{
  // = = =
```

```
class Program
{
  private const int MaxArgs = 2;

  static async Task Main(string[] args)
  {
    // Parse the command line and show help if necessary
    var parsedArgs = CommandLine.Parse(args);
    if(parsedArgs.Count == 0)
    {
      PrintHelp();
      return;
    }
    if(parsedArgs.Count > MaxArgs)
      CommandLine.ErrorExit("\nThe number of command-line arguments is incorrect." +
        "\nRun the command with no arguments to see help.");

    // Get the application arguments from the parsed list
    var groupName = CommandLine.GetArgument(parsedArgs, null, "-g", "--group-name");
    var vpcID = CommandLine.GetArgument(parsedArgs, null, "-v", "--vpc-id");
    if(string.IsNullOrEmpty(groupName))
      CommandLine.ErrorExit("\nYou must supply a name for the new group." +
        "\nRun the command with no arguments to see help.");
    if(!string.IsNullOrEmpty(vpcID) && !vpcID.StartsWith("vpc-"))
      CommandLine.ErrorExit($"\nNot a valid VPC ID: {vpcID}");

    // groupName has a value and vpcID either has a value or is null (which is fine)
    // Create the new security group and display information about it
    var securityGroups =
      await CreateSecurityGroup(new AmazonEC2Client(), groupName, vpcID);
    Console.WriteLine("Information about the security group(s):");
    foreach(var group in securityGroups)
    {
      Console.WriteLine($"\nGroupName: {group.GroupName}");
      Console.WriteLine($"GroupId: {group.GroupId}");
      Console.WriteLine($"Description: {group.Description}");
      Console.WriteLine($"VpcId (if any): {group.VpcId}");
    }
  }


  //
  // Method to create a new security group (either EC2-Classic or EC2-VPC)
  // If vpcID is empty, the security group will be for EC2-Classic
```

```
    private static async Task<List<SecurityGroup>> CreateSecurityGroup(
      IAmazonEC2 ec2Client, string groupName, string vpcID)
    {
      // See if one or more security groups with that name
      // already exist in the given VPC. If so, return the list of them.
      var securityGroups = await FindSecurityGroups(ec2Client, groupName, vpcID);
      if (securityGroups.Count > 0)
      {
        Console.WriteLine(
          $"\nOne or more security groups with name {groupName} already exist.\n");
        return securityGroups;
      }

      // If the security group doesn't already exists, create it.
      var createRequest = new CreateSecurityGroupRequest{
        GroupName = groupName
      };
      if(string.IsNullOrEmpty(vpcID))
      {
        createRequest.Description = "Security group for .NET code example (no VPC
specified)";
      }
      else
      {
        createRequest.VpcId = vpcID;
        createRequest.Description = "Security group for .NET code example (VPC: " +
vpcID + ")";
      }
      CreateSecurityGroupResponse createResponse =
        await ec2Client.CreateSecurityGroupAsync(createRequest);

      // Return the new security group
      DescribeSecurityGroupsResponse describeResponse =
        await ec2Client.DescribeSecurityGroupsAsync(new DescribeSecurityGroupsRequest{
          GroupIds = new List<string>() { createResponse.GroupId }
        });
      return describeResponse.SecurityGroups;
    }



    //
    // Method to determine if a security group with the specified name
    // already exists in the VPC
    private static async Task<List<SecurityGroup>> FindSecurityGroups(
```

```
      IAmazonEC2 ec2Client, string groupName, string vpcID)
    {
      var request = new DescribeSecurityGroupsRequest();
      request.Filters.Add(new Filter{
        Name = "group-name",
        Values = new List<string>() { groupName }
      });
      if(!string.IsNullOrEmpty(vpcID))
        request.Filters.Add(new Filter{
          Name = "vpc-id",
          Values = new List<string>() { vpcID }
        });

      var response = await ec2Client.DescribeSecurityGroupsAsync(request);
      return response.SecurityGroups;
    }


    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: EC2CreateSecGroup -g <group-name> [-v <vpc-id>]" +
        "\n  -g, --group-name: The name you would like the new security group to have."
+
        "\n  -v, --vpc-id: The ID of a VPC to which the new security group will
belong." +
        "\n     If vpc-id isn't present, the security group will be" +
        "\n     for EC2-Classic (if your AWS account supports this)" +
        "\n     or will use the default VCP for EC2-VPC.");
    }
  }


  // = = =
```

```
    //
    // Parameters:
    // - args: The command-line arguments passed into the application by the system.
    //
    // Returns:
    // A Dictionary with string Keys and Values.
    //
    // If a key is found without a matching value, Dictionary.Value is set to the key
    //  (including the dashes).
    // If a value is found without a matching key, Dictionary.Key is set to "--NoKeyN",
    //  where "N" represents sequential numbers.
    public static Dictionary<string,string> Parse(string[] args)
    {
      var parsedArgs = new Dictionary<string,string>();
      int i = 0, n = 0;
      while(i < args.Length)
      {
        // If the first argument in this iteration starts with a dash it's an option.
        if(args[i].StartsWith("-"))
        {
          var key = args[i++];
          var value = key;

          // Check to see if there's a value that goes with this option?
          if((i < args.Length) && (!args[i].StartsWith("-"))) value = args[i++];
          parsedArgs.Add(key, value);
        }

        // If the first argument in this iteration doesn't start with a dash, it's a
value
        else
        {
          parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
          n++;
        }
      }

      return parsedArgs;
    }

    //
    // Method to get an argument from the parsed command-line arguments
    //
    // Parameters:
```

```
    // - parsedArgs: The Dictionary object returned from the Parse() method (shown
 above).
    // - defaultValue: The default string to return if the specified key isn't in
 parsedArgs.
    // - keys: An array of keys to look for in parsedArgs.
    public static string GetArgument(
      Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
    {
      string retval = null;
      foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
      return retval ?? defaultReturn;
    }

    //
    // Method to exit the application with an error.
    public static void ErrorExit(string msg, int code=1)
    {
      Console.WriteLine("\nError");
      Console.WriteLine(msg);
      Environment.Exit(code);
    }
  }

}
```

Memperbarui grup keamanan

Contoh ini menunjukkan cara menggunakan aturan AWS SDK untuk .NET untuk menambahkan aturan ke grup keamanan. Secara khusus, contoh menambahkan aturan untuk mengizinkan lalu lintas masuk pada port TCP tertentu, yang dapat digunakan, misalnya, untuk koneksi jarak jauh ke instance EC2. Aplikasi mengambil ID dari grup keamanan yang ada, alamat IP (atau rentang alamat) dalam format CIDR, dan opsional nomor port TCP. Kemudian menambahkan aturan masuk ke grup keamanan yang diberikan.

> **Note**
>
> Untuk menggunakan contoh ini, Anda memerlukan alamat IP (atau rentang alamat) dalam format CIDR. Lihat Pertimbangan tambahan di akhir topik ini untuk metode mendapatkan alamat IP komputer lokal Anda.

Bagian berikut menyediakan cuplikan dari contoh ini. Kode lengkap untuk contoh ditampilkan setelah itu, dan dapat dibangun dan dijalankan apa adanya.

Topik

- Tambahkan aturan masuk
- Kode lengkap
- Pertimbangan tambahan

Tambahkan aturan masuk

Cuplikan berikut menambahkan aturan masuk ke grup keamanan untuk alamat IP tertentu (atau rentang) dan port TCP.

Contoh di akhir topik ini menunjukkan cuplikan ini digunakan.

```
//
// Method that adds a TCP ingress rule to a security group
private static async Task AddIngressRule(
  IAmazonEC2 eC2Client, string groupID, string ipAddress, int port)
{
  // Create an object to hold the request information for the rule.
  // It uses an IpPermission object to hold the IP information for the rule.
  var ingressRequest = new AuthorizeSecurityGroupIngressRequest{
    GroupId = groupID};
  ingressRequest.IpPermissions.Add(new IpPermission{
    IpProtocol = "tcp",
    FromPort = port,
    ToPort = port,
    Ipv4Ranges = new List<IpRange>() { new IpRange { CidrIp = ipAddress } }
  });

  // Create the inbound rule for the security group
  AuthorizeSecurityGroupIngressResponse responseIngress =
    await eC2Client.AuthorizeSecurityGroupIngressAsync(ingressRequest);
  Console.WriteLine($"\nNew RDP rule was written in {groupID} for {ipAddress}.");
  Console.WriteLine($"Result: {responseIngress.HttpStatusCode}");
}
```

Kode lengkap

Bagian ini menunjukkan referensi yang relevan dan kode lengkap untuk contoh ini.

Referensi SDK

NuGet paket:

- AWSSDK.EC2

Elemen pemrograman:

- Namespace Amazon.EC2

  EC2Klien Kelas Amazon
- Namespace Amazon.EC2.Model

  Kelas AuthorizeSecurityGroupIngressRequest

  Kelas AuthorizeSecurityGroupIngressResponse

  Kelas IpPermission

  Kelas IpRange

Kodenya

```
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2AddRuleForRDP
{
  // = = =
```

```
      if(parsedArgs.Count == 0)
      {
        PrintHelp();
        return;
      }

      // Get the application arguments from the parsed list
      var groupID = CommandLine.GetArgument(parsedArgs, null, "-g", "--group-id");
      var ipAddress = CommandLine.GetArgument(parsedArgs, null, "-i", "--ip-address");
      var portStr = CommandLine.GetArgument(parsedArgs, DefaultPort.ToString(), "-p",
"--port");
      if(string.IsNullOrEmpty(ipAddress))
        CommandLine.ErrorExit("\nYou must supply an IP address in CIDR format.");
      if(string.IsNullOrEmpty(groupID) || !groupID.StartsWith("sg-"))
        CommandLine.ErrorExit("\nThe ID for a security group is missing or
incorrect.");
      if(int.Parse(portStr) == 0)
        CommandLine.ErrorExit($"\nThe given TCP port number, {portStr}, isn't
allowed.");

      // Add a rule to the given security group that allows
      // inbound traffic on a TCP port
      await AddIngressRule(
        new AmazonEC2Client(), groupID, ipAddress, int.Parse(portStr));
    }


    //
    // Method that adds a TCP ingress rule to a security group
    private static async Task AddIngressRule(
      IAmazonEC2 eC2Client, string groupID, string ipAddress, int port)
    {
      // Create an object to hold the request information for the rule.
      // It uses an IpPermission object to hold the IP information for the rule.
      var ingressRequest = new AuthorizeSecurityGroupIngressRequest{
        GroupId = groupID};
      ingressRequest.IpPermissions.Add(new IpPermission{
        IpProtocol = "tcp",
        FromPort = port,
        ToPort = port,
        Ipv4Ranges = new List<IpRange>() { new IpRange { CidrIp = ipAddress } }
      });

      // Create the inbound rule for the security group
```

```
      AuthorizeSecurityGroupIngressResponse responseIngress =
        await eC2Client.AuthorizeSecurityGroupIngressAsync(ingressRequest);
      Console.WriteLine($"\nNew RDP rule was written in {groupID} for {ipAddress}.");
      Console.WriteLine($"Result: {responseIngress.HttpStatusCode}");
    }


    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: EC2AddRuleForRDP -g <group-id> -i <ip-address> [-p <port>]" +
        "\n  -g, --group-id: The ID of the security group to which you want to add the
inbound rule." +
        "\n  -i, --ip-address: An IP address or address range in CIDR format." +
        "\n  -p, --port: The TCP port number. Defaults to 3389.");
    }
  }


  // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = =
  // Class that represents a command line on the console or terminal.
  // (This is the same for all examples. When you have seen it once, you can ignore
it.)
  static class CommandLine
  {
    //
    // Method to parse a command line of the form: "--key value" or "-k value".
    //
    // Parameters:
    // - args: The command-line arguments passed into the application by the system.
    //
    // Returns:
    // A Dictionary with string Keys and Values.
    //
    // If a key is found without a matching value, Dictionary.Value is set to the key
    //  (including the dashes).
    // If a value is found without a matching key, Dictionary.Key is set to "--NoKeyN",
    //  where "N" represents sequential numbers.
    public static Dictionary<string,string> Parse(string[] args)
    {
      var parsedArgs = new Dictionary<string,string>();
```

```
      int i = 0, n = 0;
      while(i < args.Length)
      {
        // If the first argument in this iteration starts with a dash it's an option.
        if(args[i].StartsWith("-"))
        {
          var key = args[i++];
          var value = key;

          // Check to see if there's a value that goes with this option?
          if((i < args.Length) && (!args[i].StartsWith("-"))) value = args[i++];
          parsedArgs.Add(key, value);
        }

        // If the first argument in this iteration doesn't start with a dash, it's a
value
        else
        {
          parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
          n++;
        }
      }

      return parsedArgs;
    }

    //
    // Method to get an argument from the parsed command-line arguments
    //
    // Parameters:
    // - parsedArgs: The Dictionary object returned from the Parse() method (shown
above).
    // - defaultValue: The default string to return if the specified key isn't in
parsedArgs.
    // - keys: An array of keys to look for in parsedArgs.
    public static string GetArgument(
      Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
    {
      string retval = null;
      foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
      return retval ?? defaultReturn;
    }
```

```
    //
    // Method to exit the application with an error.
    public static void ErrorExit(string msg, int code=1)
    {
      Console.WriteLine("\nError");
      Console.WriteLine(msg);
      Environment.Exit(code);
    }
  }

}
```

Pertimbangan tambahan

- Jika Anda tidak menyediakan nomor port, aplikasi default ke port 3389. Ini adalah port untuk Windows RDP, yang memungkinkan Anda untuk terhubung ke instans EC2 yang menjalankan Windows. Jika Anda meluncurkan instans EC2 yang menjalankan Linux, Anda dapat menggunakan port TCP 22 (SSH) sebagai gantinya.

- Perhatikan bahwa contoh disetel `IpProtocol` ke “tcp”. Nilai untuk `IpProtocol` dapat ditemukan dalam deskripsi untuk `IpProtocol` properti IpPermissionkelas.

- Anda mungkin menginginkan alamat IP komputer lokal Anda saat menggunakan contoh ini. Berikut ini adalah beberapa cara di mana Anda bisa mendapatkan alamat.
  - Jika komputer lokal Anda (dari mana Anda akan terhubung ke instans EC2 Anda) memiliki alamat IP publik statis, Anda dapat menggunakan layanan untuk mendapatkan alamat itu. Salah satu layanan tersebut adalah http://checkip.amazonaws.com/. Untuk membaca selengkapnya tentang mengotorisasi lalu lintas masuk, lihat Menambahkan aturan ke grup keamanan dan Aturan grup keamanan untuk kasus penggunaan yang berbeda di Panduan Pengguna Amazon EC2.
  - Cara lain untuk mendapatkan alamat IP komputer lokal Anda adalah dengan menggunakan konsol Amazon EC2.

    Pilih salah satu grup keamanan Anda, pilih tab Aturan masuk, dan pilih Edit aturan masuk. Dalam aturan masuk, buka menu tarik-turun di kolom Sumber dan pilih IP Saya untuk melihat alamat IP komputer lokal Anda dalam format CIDR. Pastikan untuk Membatalkan operasi.

- Anda dapat memverifikasi hasil contoh ini dengan memeriksa daftar grup keamanan di konsol Amazon EC2.

Bekerja dengan pasangan kunci Amazon EC2

Amazon EC2 menggunakan kriptografi kunci publik untuk mengenkripsi dan mendekripsi informasi login. Kriptografi kunci publik menggunakan kunci publik untuk mengenkripsi data, dan kemudian penerima menggunakan kunci pribadi untuk mendekripsi data. Kunci publik dan privat dikenal sebagai pasangan kunci. Jika Anda ingin masuk ke instans EC2, Anda harus menentukan key pair ketika Anda meluncurkannya, dan kemudian memberikan kunci pribadi pasangan saat Anda terhubung dengannya.

Saat meluncurkan instans EC2, Anda dapat membuat key pair untuknya atau menggunakan salah satu yang sudah Anda gunakan saat meluncurkan instance lain. Untuk membaca selengkapnya tentang pasangan kunci Amazon EC2, lihat Bekerja dengan pasangan kunci Amazon EC2 di Panduan Pengguna Amazon EC2.

Untuk informasi tentang APIs dan prasyarat, lihat bagian induk (). Bekerja dengan Amazon EC2

Topik

- Membuat dan menampilkan pasangan kunci
- Menghapus pasangan kunci

Membuat dan menampilkan pasangan kunci

Contoh ini menunjukkan kepada Anda cara menggunakan AWS SDK untuk .NET to create a key pair. Aplikasi ini mengambil nama untuk key pair baru dan nama file PEM (dengan ekstensi “.pem”). Ini menciptakan keypair, menulis kunci pribadi ke file PEM, dan kemudian menampilkan semua pasangan kunci yang tersedia. Jika Anda tidak memberikan argumen baris perintah, aplikasi hanya menampilkan semua pasangan kunci yang tersedia.

Bagian berikut menyediakan cuplikan dari contoh ini. Kode lengkap untuk contoh ditampilkan setelah itu, dan dapat dibangun dan dijalankan apa adanya.

Topik

- Buat pasangan kunci
- Tampilkan pasangan kunci yang tersedia
- Kode lengkap

- Pertimbangan tambahan

Buat pasangan kunci

Cuplikan berikut membuat key pair dan kemudian menyimpan kunci pribadi ke file PEM yang diberikan.

Contoh di akhir topik ini menunjukkan cuplikan ini digunakan.

```
//
// Method to create a key pair and save the key material in a PEM file
private static async Task CreateKeyPair(
  IAmazonEC2 ec2Client, string keyPairName, string pemFileName)
{
  // Create the key pair
  CreateKeyPairResponse response =
    await ec2Client.CreateKeyPairAsync(new CreateKeyPairRequest{
      KeyName = keyPairName
    });
  Console.WriteLine($"\nCreated new key pair: {response.KeyPair.KeyName}");

  // Save the private key in a PEM file
  using (var s = new FileStream(pemFileName, FileMode.Create))
  using (var writer = new StreamWriter(s))
  {
    writer.WriteLine(response.KeyPair.KeyMaterial);
  }
}
```

Tampilkan pasangan kunci yang tersedia

Cuplikan berikut menampilkan daftar pasangan kunci yang tersedia.

Contoh di akhir topik ini menunjukkan cuplikan ini digunakan.

```
//
// Method to show the key pairs that are available
private static async Task EnumerateKeyPairs(IAmazonEC2 ec2Client)
{
  DescribeKeyPairsResponse response = await ec2Client.DescribeKeyPairsAsync();
  Console.WriteLine("Available key pairs:");
  foreach (KeyPairInfo item in response.KeyPairs)
    Console.WriteLine($"  {item.KeyName}");
```

```
    }
```

Kode lengkap

Bagian ini menunjukkan referensi yang relevan dan kode lengkap untuk contoh ini.

Referensi SDK

NuGet paket:

- AWSSDK.EC2

Elemen pemrograman:

- Namespace Amazon.EC2

  EC2Klien Kelas Amazon
- Namespace Amazon.EC2.Model

  Kelas CreateKeyPairRequest

  Kelas CreateKeyPairResponse

  Kelas DescribeKeyPairsResponse

  Kelas KeyPairInfo

Kodenya

```
using System;
using System.Threading.Tasks;
using System.IO;
using Amazon.EC2;
using Amazon.EC2.Model;
using System.Collections.Generic;

namespace EC2CreateKeyPair
{
  // = = =
```

```
{
  static async Task Main(string[] args)
  {
    // Create the EC2 client
    var ec2Client = new AmazonEC2Client();

    // Parse the command line and show help if necessary
    var parsedArgs = CommandLine.Parse(args);
    if(parsedArgs.Count == 0)
    {
      // In the case of no command-line arguments,
      // just show help and the existing key pairs
      PrintHelp();
      Console.WriteLine("\nNo arguments specified.");
      Console.Write(
        "Do you want to see a list of the existing key pairs? ((y) or n): ");
      string response = Console.ReadLine();
      if((string.IsNullOrEmpty(response)) || (response.ToLower() == "y"))
        await EnumerateKeyPairs(ec2Client);
      return;
    }

    // Get the application arguments from the parsed list
    string keyPairName =
      CommandLine.GetArgument(parsedArgs, null, "-k", "--keypair-name");
    string pemFileName =
      CommandLine.GetArgument(parsedArgs, null, "-p", "--pem-filename");
    if(string.IsNullOrEmpty(keyPairName))
      CommandLine.ErrorExit("\nNo key pair name specified." +
        "\nRun the command with no arguments to see help.");
    if(string.IsNullOrEmpty(pemFileName) || !pemFileName.EndsWith(".pem"))
      CommandLine.ErrorExit("\nThe PEM filename is missing or incorrect." +
        "\nRun the command with no arguments to see help.");

    // Create the key pair
    await CreateKeyPair(ec2Client, keyPairName, pemFileName);
    await EnumerateKeyPairs(ec2Client);
  }


  //
  // Method to create a key pair and save the key material in a PEM file
  private static async Task CreateKeyPair(
    IAmazonEC2 ec2Client, string keyPairName, string pemFileName)
```

```
    {
      // Create the key pair
      CreateKeyPairResponse response =
        await ec2Client.CreateKeyPairAsync(new CreateKeyPairRequest{
          KeyName = keyPairName
        });
      Console.WriteLine($"\nCreated new key pair: {response.KeyPair.KeyName}");

      // Save the private key in a PEM file
      using (var s = new FileStream(pemFileName, FileMode.Create))
      using (var writer = new StreamWriter(s))
      {
        writer.WriteLine(response.KeyPair.KeyMaterial);
      }
    }


    //
    // Method to show the key pairs that are available
    private static async Task EnumerateKeyPairs(IAmazonEC2 ec2Client)
    {
      DescribeKeyPairsResponse response = await ec2Client.DescribeKeyPairsAsync();
      Console.WriteLine("Available key pairs:");
      foreach (KeyPairInfo item in response.KeyPairs)
        Console.WriteLine($"  {item.KeyName}");
    }


    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: EC2CreateKeyPair -k <keypair-name> -p <pem-filename>" +
        "\n  -k, --keypair-name: The name you want to assign to the key pair." +
        "\n  -p, --pem-filename: The name of the PEM file to create, with a \".pem\"
extension.");
    }
  }


  // = = =
```

```
 // (This is the same for all examples. When you have seen it once, you can ignore
it.)
 static class CommandLine
 {
   //
   // Method to parse a command line of the form: "--key value" or "-k value".
   //
   // Parameters:
   // - args: The command-line arguments passed into the application by the system.
   //
   // Returns:
   // A Dictionary with string Keys and Values.
   //
   // If a key is found without a matching value, Dictionary.Value is set to the key
   //  (including the dashes).
   // If a value is found without a matching key, Dictionary.Key is set to "--NoKeyN",
   //  where "N" represents sequential numbers.
   public static Dictionary<string,string> Parse(string[] args)
   {
     var parsedArgs = new Dictionary<string,string>();
     int i = 0, n = 0;
     while(i < args.Length)
     {
       // If the first argument in this iteration starts with a dash it's an option.
       if(args[i].StartsWith("-"))
       {
         var key = args[i++];
         var value = key;

         // Check to see if there's a value that goes with this option?
         if((i < args.Length) && (!args[i].StartsWith("-"))) value = args[i++];
         parsedArgs.Add(key, value);
       }

       // If the first argument in this iteration doesn't start with a dash, it's a
value
       else
       {
         parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
         n++;
       }
     }

     return parsedArgs;
```

```
    }

    //
    // Method to get an argument from the parsed command-line arguments
    //
    // Parameters:
    // - parsedArgs: The Dictionary object returned from the Parse() method (shown
 above).
    // - defaultValue: The default string to return if the specified key isn't in
 parsedArgs.
    // - keys: An array of keys to look for in parsedArgs.
    public static string GetArgument(
      Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
    {
      string retval = null;
      foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
      return retval ?? defaultReturn;
    }

    //
    // Method to exit the application with an error.
    public static void ErrorExit(string msg, int code=1)
    {
      Console.WriteLine("\nError");
      Console.WriteLine(msg);
      Environment.Exit(code);
    }
  }

}
```

Pertimbangan tambahan

- Setelah Anda menjalankan contoh, Anda dapat melihat key pair baru di konsol Amazon EC2.

- Ketika Anda membuat key pair, Anda harus menyimpan kunci pribadi yang dikembalikan karena Anda tidak dapat mengambil kunci pribadi nanti.

Menghapus pasangan kunci

Contoh ini menunjukkan kepada Anda cara menggunakan AWS SDK untuk .NET to delete a key pair. Aplikasi ini mengambil nama sebuah key pair. Ini menghapus key pair dan kemudian menampilkan semua pasangan kunci yang tersedia. Jika Anda tidak memberikan argumen baris perintah, aplikasi hanya menampilkan semua pasangan kunci yang tersedia.

Bagian berikut menyediakan cuplikan dari contoh ini. Kode lengkap untuk contoh ditampilkan setelah itu, dan dapat dibangun dan dijalankan apa adanya.

Topik

- Hapus key pair
- Tampilkan pasangan kunci yang tersedia
- Kode lengkap

Hapus key pair

Cuplikan berikut menghapus key pair.

Contoh di akhir topik ini menunjukkan cuplikan ini digunakan.

```
//
// Method to delete a key pair
private static async Task DeleteKeyPair(IAmazonEC2 ec2Client, string keyName)
{
  await ec2Client.DeleteKeyPairAsync(new DeleteKeyPairRequest{
    KeyName = keyName});
  Console.WriteLine($"\nKey pair {keyName} has been deleted (if it existed).");
}
```

Tampilkan pasangan kunci yang tersedia

Cuplikan berikut menampilkan daftar pasangan kunci yang tersedia.

Contoh di akhir topik ini menunjukkan cuplikan ini digunakan.

```
//
// Method to show the key pairs that are available
private static async Task EnumerateKeyPairs(IAmazonEC2 ec2Client)
{
  DescribeKeyPairsResponse response = await ec2Client.DescribeKeyPairsAsync();
```

```
      Console.WriteLine("Available key pairs:");
      foreach (KeyPairInfo item in response.KeyPairs)
        Console.WriteLine($"  {item.KeyName}");
    }
```

Kode lengkap

Bagian ini menunjukkan referensi yang relevan dan kode lengkap untuk contoh ini.

Referensi SDK

NuGet paket:

- AWSSDK.EC2

Elemen pemrograman:

- Namespace Amazon.EC2

  EC2Klien Kelas Amazon
- Namespace Amazon.EC2.Model

  Kelas DeleteKeyPairRequest

  Kelas DescribeKeyPairsResponse

  Kelas KeyPairInfo

Kodenya

```
using System;
using System.Threading.Tasks;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2DeleteKeyPair
{
  class Program
  {
    static async Task Main(string[] args)
    {
      // Create the EC2 client
```

```
      var ec2Client = new AmazonEC2Client();

      if(args.Length == 1)
      {
        // Delete a key pair (if it exists)
        await DeleteKeyPair(ec2Client, args[0]);

        // Display the key pairs that are left
        await EnumerateKeyPairs(ec2Client);
      }
      else
      {
        Console.WriteLine("\nUsage: EC2DeleteKeyPair keypair-name");
        Console.WriteLine("  keypair-name - The name of the key pair you want to
delete.");
        Console.WriteLine("\nNo arguments specified.");
        Console.Write(
          "Do you want to see a list of the existing key pairs? ((y) or n): ");
        string response = Console.ReadLine();
        if((string.IsNullOrEmpty(response)) || (response.ToLower() == "y"))
          await EnumerateKeyPairs(ec2Client);
      }
    }


    //
    // Method to delete a key pair
    private static async Task DeleteKeyPair(IAmazonEC2 ec2Client, string keyName)
    {
      await ec2Client.DeleteKeyPairAsync(new DeleteKeyPairRequest{
        KeyName = keyName});
      Console.WriteLine($"\nKey pair {keyName} has been deleted (if it existed).");
    }


    //
    // Method to show the key pairs that are available
    private static async Task EnumerateKeyPairs(IAmazonEC2 ec2Client)
    {
      DescribeKeyPairsResponse response = await ec2Client.DescribeKeyPairsAsync();
      Console.WriteLine("Available key pairs:");
      foreach (KeyPairInfo item in response.KeyPairs)
        Console.WriteLine($"  {item.KeyName}");
    }
```

```
    }
}
```

Melihat Wilayah Amazon EC2 dan Availability Zone

Amazon EC2 di-host di beberapa lokasi di seluruh dunia. Lokasi ini terdiri dari Wilayah dan Zona Ketersediaan . Setiap Wilayah adalah wilayah geografis terpisah yang memiliki beberapa lokasi terisolasi yang dikenal sebagai Availability Zone.

Untuk membaca lebih lanjut tentang Wilayah dan Availability Zone, lihat Wilayah dan Zona di Panduan Pengguna Amazon EC2.

Contoh ini menunjukkan kepada Anda cara menggunakan AWS SDK untuk .NET untuk mendapatkan detail tentang Wilayah dan Availability Zone yang terkait dengan klien EC2. Aplikasi ini menampilkan daftar Wilayah dan Availability Zone yang tersedia untuk klien EC2.

Referensi SDK

NuGet paket:

- AWSSDK.EC2

Elemen pemrograman:

- Namespace Amazon.EC2

  EC2Klien Kelas Amazon
- Namespace Amazon.EC2.Model

  Kelas DescribeAvailabilityZonesResponse

  Kelas DescribeRegionsResponse

  Kelas AvailabilityZone

  Wilayah Kelas

```
using System;
using System.Threading.Tasks;
using Amazon.EC2;
using Amazon.EC2.Model;
```

```
namespace EC2RegionsAndZones
{
  class Program
  {
    static async Task Main(string[] args)
    {
      Console.WriteLine(
        "Finding the Regions and Availability Zones available to an EC2 client...");

      // Create the EC2 client
      var ec2Client = new AmazonEC2Client();

      // Display the Regions and Availability Zones
      await DescribeRegions(ec2Client);
      await DescribeAvailabilityZones(ec2Client);
    }


    //
    // Method to display Regions
    private static async Task DescribeRegions(IAmazonEC2 ec2Client)
    {
      Console.WriteLine("\nRegions that are enabled for the EC2 client:");
      DescribeRegionsResponse response = await ec2Client.DescribeRegionsAsync();
      foreach (Region region in response.Regions)
        Console.WriteLine(region.RegionName);
    }


    //
    // Method to display Availability Zones
    private static async Task DescribeAvailabilityZones(IAmazonEC2 ec2Client)
    {
      Console.WriteLine("\nAvailability Zones for the EC2 client's region:");
      DescribeAvailabilityZonesResponse response =
        await ec2Client.DescribeAvailabilityZonesAsync();
      foreach (AvailabilityZone az in response.AvailabilityZones)
        Console.WriteLine(az.ZoneName);
    }
  }
}
```

Bekerja dengan instans Amazon EC2

Anda dapat menggunakan AWS SDK untuk .NET untuk mengontrol instans Amazon EC2 dengan operasi seperti membuat, memulai, dan mengakhiri. Topik di bagian ini memberikan beberapa contoh bagaimana melakukan ini. Untuk membaca selengkapnya tentang instans EC2, lihat instans Amazon EC2 di Panduan Pengguna Amazon EC2.

Untuk informasi tentang APIs dan prasyarat, lihat bagian induk (). Bekerja dengan Amazon EC2

Topik

- Meluncurkan instans Amazon EC2
- Mengakhiri instans Amazon EC2

Meluncurkan instans Amazon EC2

Contoh ini menunjukkan kepada Anda cara menggunakan AWS SDK untuk .NET untuk meluncurkan satu atau lebih instans Amazon EC2 yang dikonfigurasi secara identik dari Amazon Machine Image (AMI) yang sama. Menggunakan beberapa input yang Anda berikan, aplikasi meluncurkan instans EC2 dan kemudian memonitor instance hingga keluar dari status “Tertunda”.

Saat instans EC2 Anda berjalan, Anda dapat menghubungkannya dari jarak jauh, seperti yang dijelaskan dalam. (opsional) Connect ke instance

> **Warning**
>
> EC2-Classic sudah tidak digunakan lagi pada 15 Agustus 2022. Kami menyarankan Anda bermigrasi dari EC2-Classic ke VPC. Untuk informasi lebih lanjut, lihat posting blog EC2-Classic Networking is Retiring - Inilah Cara Mempersiapkan.

Bagian berikut menyediakan cuplikan dan informasi lain untuk contoh ini. Kode lengkap untuk contoh ditampilkan setelah cuplikan, dan dapat dibangun dan dijalankan apa adanya.

Topik

- Kumpulkan apa yang Anda butuhkan
- Luncurkan sebuah instans
- Pantau instance
- Kode lengkap

- Pertimbangan tambahan
- (opsional) Connect ke instance
- Bersihkan

Kumpulkan apa yang Anda butuhkan

Untuk meluncurkan instans EC2, Anda memerlukan beberapa hal.

- VPC tempat instance akan diluncurkan. Jika itu adalah instance Windows dan Anda akan menghubungkannya melalui RDP, VPC kemungkinan besar harus memiliki gateway internet yang terpasang padanya, serta entri untuk gateway internet di tabel rute. Untuk informasi lebih lanjut, lihat Gateway internet di Panduan Pengguna Amazon VPC.

- ID subnet yang ada di VPC tempat instance akan diluncurkan. Cara mudah untuk menemukan atau membuat ini adalah dengan masuk ke konsol VPC Amazon, tetapi Anda juga dapat memperolehnya secara terprogram dengan menggunakan metode dan. CreateSubnetAsyncDescribeSubnetsAsync

  > **Note**
  >
  > Jika Anda tidak menyediakan parameter ini, instance baru akan diluncurkan di VPC default untuk akun Anda.

- ID grup keamanan yang ada milik VPC tempat instance akan diluncurkan. Untuk informasi selengkapnya, lihat Bekerja dengan grup keamanan di Amazon EC2.

- Jika Anda ingin terhubung ke instance baru, grup keamanan yang disebutkan sebelumnya harus memiliki aturan masuk yang sesuai yang memungkinkan lalu lintas SSH pada port 22 (instance Linux) atau lalu lintas RDP pada port 3389 (instance Windows). Untuk informasi tentang cara melakukan iniMemperbarui grup keamanan, lihat, termasuk Pertimbangan tambahan mendekati akhir topik itu.

- Amazon Machine Image (AMI) yang akan digunakan untuk membuat instance. Untuk selengkapnya AMIs, lihat Gambar Mesin Amazon (AMIs) di Panduan Pengguna Amazon EC2. Secara khusus, lihat Menemukan AMI dan Dibagikan AMIs.

- Nama key pair EC2 yang ada, yang digunakan untuk terhubung ke instance baru. Untuk informasi selengkapnya, lihat Bekerja dengan pasangan kunci Amazon EC2.

- Nama file PEM yang berisi kunci pribadi dari key pair EC2 yang disebutkan sebelumnya. File PEM digunakan saat Anda terhubung dari jarak jauh ke instance.

Luncurkan sebuah instans

Cuplikan berikut meluncurkan instans EC2.

Contoh di dekat akhir topik ini menunjukkan cuplikan ini digunakan.

```
//
// Method to launch the instances
// Returns a list with the launched instance IDs
private static async Task<List<string>> LaunchInstances(
  IAmazonEC2 ec2Client, RunInstancesRequest requestLaunch)
{
  var instanceIds = new List<string>();
  RunInstancesResponse responseLaunch =
    await ec2Client.RunInstancesAsync(requestLaunch);

  Console.WriteLine("\nNew instances have been created.");
  foreach (Instance item in responseLaunch.Reservation.Instances)
  {
    instanceIds.Add(item.InstanceId);
    Console.WriteLine($"  New instance: {item.InstanceId}");
  }

  return instanceIds;
}
```

Pantau instance

Cuplikan berikut memonitor instance hingga keluar dari status “Tertunda”.

Contoh di dekat akhir topik ini menunjukkan cuplikan ini digunakan.

Lihat InstanceStatekelas untuk nilai valid dari `Instance.State.Code` properti.

```
//
// Method to wait until the instances are running (or at least not pending)
```

```
    private static async Task CheckState(IAmazonEC2 ec2Client, List<string>
instanceIds)
    {
      Console.WriteLine(
        "\nWaiting for the instances to start." +
        "\nPress any key to stop waiting. (Response might be slightly delayed.)");

      int numberRunning;
      DescribeInstancesResponse responseDescribe;
      var requestDescribe = new DescribeInstancesRequest{
        InstanceIds = instanceIds};

      // Check every couple of seconds
      int wait = 2000;
      while(true)
      {
        // Get and check the status for each of the instances to see if it's past
pending.
        // Once all instances are past pending, break out.
        // (For this example, we are assuming that there is only one reservation.)
        Console.Write(".");
        numberRunning = 0;
        responseDescribe = await ec2Client.DescribeInstancesAsync(requestDescribe);
        foreach(Instance i in responseDescribe.Reservations[0].Instances)
        {
          // Check the lower byte of State.Code property
          // Code == 0 is the pending state
          if((i.State.Code & 255) > 0) numberRunning++;
        }
        if(numberRunning == responseDescribe.Reservations[0].Instances.Count)
          break;

        // Wait a bit and try again (unless the user wants to stop waiting)
        Thread.Sleep(wait);
        if(Console.KeyAvailable)
          break;
      }

      Console.WriteLine("\nNo more instances are pending.");
      foreach(Instance i in responseDescribe.Reservations[0].Instances)
      {
        Console.WriteLine($"For {i.InstanceId}:");
        Console.WriteLine($"  VPC ID: {i.VpcId}");
        Console.WriteLine($"  Instance state: {i.State.Name}");
```

```
        Console.WriteLine($"  Public IP address: {i.PublicIpAddress}");
        Console.WriteLine($"  Public DNS name: {i.PublicDnsName}");
        Console.WriteLine($"  Key pair name: {i.KeyName}");
      }
    }
```

Kode lengkap

Bagian ini menunjukkan referensi yang relevan dan kode lengkap untuk contoh ini.

Referensi SDK

NuGet paket:

- AWSSDK.EC2

Elemen pemrograman:

- Namespace Amazon.EC2

  EC2Klien Kelas Amazon

  Kelas InstanceType
- Namespace Amazon.EC2.Model

  Kelas DescribeInstancesRequest

  Kelas DescribeInstancesResponse

  Instance Kelas

  Kelas InstanceNetworkInterfaceSpecification

  Kelas RunInstancesRequest

  Kelas RunInstancesResponse

Kodenya

```
using System;
using System.Threading;
using System.Threading.Tasks;
```

```
using System.Collections.Generic;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2LaunchInstance
{
  // = = =
```

```
      // Launch the instances and wait for them to start running
      var instanceIds = await LaunchInstances(ec2Client, request);
      await CheckState(ec2Client, instanceIds);
    }


    //
    // Method to put together the properties needed to launch the instance.
    private static RunInstancesRequest GetRequestData(
      string groupID, string ami, string keyPairName, string subnetID)
    {
      // Common properties
      var groupIDs = new List<string>() { groupID };
      var request = new RunInstancesRequest()
      {
        // The first three of these would be additional command-line arguments or
similar.
        InstanceType = InstanceType.T1Micro,
        MinCount = 1,
        MaxCount = 1,
        ImageId = ami,
        KeyName = keyPairName
      };

      // Properties specifically for EC2 in a VPC.
      if(!string.IsNullOrEmpty(subnetID))
      {
        request.NetworkInterfaces =
          new List<InstanceNetworkInterfaceSpecification>() {
            new InstanceNetworkInterfaceSpecification() {
              DeviceIndex = 0,
              SubnetId = subnetID,
              Groups = groupIDs,
              AssociatePublicIpAddress = true
            }
          };
      }

      // Properties specifically for EC2-Classic
      else
      {
        request.SecurityGroupIds = groupIDs;
      }
```

```
      return request;
    }


    //
    // Method to launch the instances
    // Returns a list with the launched instance IDs
    private static async Task<List<string>> LaunchInstances(
      IAmazonEC2 ec2Client, RunInstancesRequest requestLaunch)
    {
      var instanceIds = new List<string>();
      RunInstancesResponse responseLaunch =
        await ec2Client.RunInstancesAsync(requestLaunch);

      Console.WriteLine("\nNew instances have been created.");
      foreach (Instance item in responseLaunch.Reservation.Instances)
      {
        instanceIds.Add(item.InstanceId);
        Console.WriteLine($"  New instance: {item.InstanceId}");
      }

      return instanceIds;
    }


    //
    // Method to wait until the instances are running (or at least not pending)
    private static async Task CheckState(IAmazonEC2 ec2Client, List<string>
instanceIds)
    {
      Console.WriteLine(
        "\nWaiting for the instances to start." +
        "\nPress any key to stop waiting. (Response might be slightly delayed.)");

      int numberRunning;
      DescribeInstancesResponse responseDescribe;
      var requestDescribe = new DescribeInstancesRequest{
        InstanceIds = instanceIds};

      // Check every couple of seconds
      int wait = 2000;
      while(true)
      {
```

```
        // Get and check the status for each of the instances to see if it's past
pending.
        // Once all instances are past pending, break out.
        // (For this example, we are assuming that there is only one reservation.)
        Console.Write(".");
        numberRunning = 0;
        responseDescribe = await ec2Client.DescribeInstancesAsync(requestDescribe);
        foreach(Instance i in responseDescribe.Reservations[0].Instances)
        {
          // Check the lower byte of State.Code property
          // Code == 0 is the pending state
          if((i.State.Code & 255) > 0) numberRunning++;
        }
        if(numberRunning == responseDescribe.Reservations[0].Instances.Count)
          break;

        // Wait a bit and try again (unless the user wants to stop waiting)
        Thread.Sleep(wait);
        if(Console.KeyAvailable)
          break;
      }

      Console.WriteLine("\nNo more instances are pending.");
      foreach(Instance i in responseDescribe.Reservations[0].Instances)
      {
        Console.WriteLine($"For {i.InstanceId}:");
        Console.WriteLine($"  VPC ID: {i.VpcId}");
        Console.WriteLine($"  Instance state: {i.State.Name}");
        Console.WriteLine($"  Public IP address: {i.PublicIpAddress}");
        Console.WriteLine($"  Public DNS name: {i.PublicDnsName}");
        Console.WriteLine($"  Key pair name: {i.KeyName}");
      }
    }


    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: EC2LaunchInstance -g <group-id> -a <ami-id> -k <keypair-name> [-s
<subnet-id>]" +
        "\n  -g, --group-id: The ID of the security group." +
        "\n  -a, --ami-id: The ID of an Amazon Machine Image." +
```

```
        "\n  -k, --keypair-name - The name of a key pair." +
        "\n  -s, --subnet-id: The ID of a subnet. Required only for EC2 in a VPC.");
    }
  }


  // = = =
```

```
        // If the first argument in this iteration doesn't start with a dash, it's a
 value
        else
        {
          parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
          n++;
        }
      }

      return parsedArgs;
    }

    //
    // Method to get an argument from the parsed command-line arguments
    //
    // Parameters:
    // - parsedArgs: The Dictionary object returned from the Parse() method (shown
 above).
    // - defaultValue: The default string to return if the specified key isn't in
 parsedArgs.
    // - keys: An array of keys to look for in parsedArgs.
    public static string GetArgument(
      Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
    {
      string retval = null;
      foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
      return retval ?? defaultReturn;
    }

    //
    // Method to exit the application with an error.
    public static void ErrorExit(string msg, int code=1)
    {
      Console.WriteLine("\nError");
      Console.WriteLine(msg);
      Environment.Exit(code);
    }
  }

}
```

Pertimbangan tambahan

- Saat memeriksa status instans EC2, Anda dapat menambahkan filter ke `Filter` properti DescribeInstancesRequestobjek. Dengan menggunakan teknik ini, Anda dapat membatasi permintaan ke instance tertentu; misalnya, instance dengan tag tertentu yang ditentukan pengguna.

- Untuk singkatnya, beberapa properti diberi nilai tipikal. Salah satu atau semua properti ini dapat ditentukan secara terprogram atau dengan masukan pengguna.

- Nilai yang dapat Anda gunakan untuk `MaxCount` properti `MinCount` dan RunInstancesRequestobjek ditentukan oleh Zona Ketersediaan target dan jumlah maksimum instance yang diizinkan untuk jenis instans. Untuk informasi selengkapnya, lihat Berapa banyak instans yang dapat saya jalankan di Amazon EC2 di FAQ Umum Amazon EC2.

- Jika Anda ingin menggunakan jenis instance yang berbeda dari contoh ini, ada beberapa jenis instance untuk dipilih. Untuk informasi selengkapnya, lihat jenis instans Amazon EC2 di Panduan Pengguna Amazon EC2. Lihat juga Detail Jenis Instance dan Penjelajah Jenis Instance.

- Anda juga dapat melampirkan peran IAM ke instance saat meluncurkannya. Untuk melakukannya, buat IamInstanceProfileSpecificationobjek yang `Name` propertinya disetel ke nama peran IAM. Kemudian tambahkan objek itu ke `IamInstanceProfile` properti RunInstancesRequestobjek.

  > **Note**
  >
  > Untuk meluncurkan instans EC2 yang memiliki peran IAM terpasang, konfigurasi pengguna IAM harus menyertakan izin tertentu. Untuk informasi selengkapnya tentang izin yang diperlukan, lihat Memberi izin pengguna untuk meneruskan peran IAM ke instans di Panduan Pengguna Amazon EC2.

(opsional) Connect ke instance

Setelah instance berjalan, Anda dapat menghubungkannya dari jarak jauh dengan menggunakan klien jarak jauh yang sesuai. Untuk instance Linux dan Windows, Anda memerlukan alamat IP publik instans atau nama DNS publik. Anda juga membutuhkan yang berikut ini.

Untuk instance Linux

Anda dapat menggunakan klien SSH untuk terhubung ke instance Linux Anda. Pastikan bahwa grup keamanan yang Anda gunakan saat meluncurkan instans memungkinkan lalu lintas SSH pada port 22, seperti yang dijelaskan dalamMemperbarui grup keamanan.

Anda juga memerlukan bagian pribadi dari key pair yang Anda gunakan untuk meluncurkan instance; yaitu, file PEM.

Untuk informasi selengkapnya, lihat Connect ke instans Linux Anda di Panduan Pengguna Amazon EC2.

Untuk contoh Windows

Anda dapat menggunakan klien RDP untuk terhubung ke instans Anda. Pastikan bahwa grup keamanan yang Anda gunakan saat meluncurkan instans memungkinkan lalu lintas RDP pada port 3389, seperti yang dijelaskan dalam. Memperbarui grup keamanan

Anda juga memerlukan kata sandi Administrator. Anda dapat memperoleh ini dengan menggunakan kode contoh berikut, yang memerlukan ID instance dan bagian pribadi dari key pair yang digunakan untuk meluncurkan instance; yaitu, file PEM.

Untuk informasi selengkapnya, lihat Connect ke instans Windows Anda di Panduan Pengguna Amazon EC2.

> ⚠ Warning
>
> Kode contoh ini mengembalikan kata sandi Administrator plaintext untuk instance Anda.

Referensi SDK

NuGet paket:

- AWSSDK.EC2

Elemen pemrograman:

- Namespace Amazon.EC2

  EC2Klien Kelas Amazon

- Namespace Amazon.EC2.Model

  Kelas GetPasswordDataRequest

  Kelas GetPasswordDataResponse

Kodenya

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2GetWindowsPassword
{
  // = = =
```

```
      // Create the EC2 client
      var ec2Client = new AmazonEC2Client();

      // Get and display the password
      string password = await GetPassword(ec2Client, instanceID, pemFileName);
      Console.WriteLine($"\nPassword: {password}");
    }


    //
    // Method to get the administrator password of a Windows EC2 instance
    private static async Task<string> GetPassword(
      IAmazonEC2 ec2Client, string instanceID, string pemFilename)
    {
      string password = string.Empty;
      GetPasswordDataResponse response =
        await ec2Client.GetPasswordDataAsync(new GetPasswordDataRequest{
          InstanceId = instanceID});
      if(response.PasswordData != null)
      {
        password = response.GetDecryptedPassword(File.ReadAllText(pemFilename));
      }
      else
      {
        Console.WriteLine($"\nThe password is not available for instance
{instanceID}.");
        Console.WriteLine($"If this is a Windows instance, the password might not be
ready.");
      }
      return password;
    }


    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: EC2GetWindowsPassword -i <instance-id> -p pem-filename" +
        "\n  -i, --instance-id: The name of the EC2 instance." +
        "\n  -p, --pem-filename: The name of the PEM file with the private key.");
    }
  }
```

```
 // = = =
```

```
      }

      return parsedArgs;
    }

    //
    // Method to get an argument from the parsed command-line arguments
    //
    // Parameters:
    // - parsedArgs: The Dictionary object returned from the Parse() method (shown
 above).
    // - defaultValue: The default string to return if the specified key isn't in
 parsedArgs.
    // - keys: An array of keys to look for in parsedArgs.
    public static string GetArgument(
      Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
    {
      string retval = null;
      foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
      return retval ?? defaultReturn;
    }

    //
    // Method to exit the application with an error.
    public static void ErrorExit(string msg, int code=1)
    {
      Console.WriteLine("\nError");
      Console.WriteLine(msg);
      Environment.Exit(code);
    }
  }

}
```

Bersihkan

Ketika Anda tidak lagi membutuhkan instans EC2 Anda, pastikan untuk menghentikannya, seperti yang dijelaskan dalam. Mengakhiri instans Amazon EC2

Mengakhiri instans Amazon EC2

Jika Anda tidak lagi membutuhkan satu atau lebih instans Amazon EC2, Anda dapat menghentikannya.

Contoh ini menunjukkan cara menggunakan instans AWS SDK untuk .NET untuk mengakhiri EC2. Dibutuhkan contoh ID sebagai input.

Referensi SDK

NuGet paket:

- AWSSDK.EC2

Elemen pemrograman:

- Namespace Amazon.EC2

  EC2Klien Kelas Amazon
- Namespace Amazon.EC2.Model

  Kelas TerminateInstancesRequest

  Kelas TerminateInstancesResponse

```
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2TerminateInstance
{
  class Program
  {
    static async Task Main(string[] args)
    {
      if((args.Length == 1) && (args[0].StartsWith("i-")))
      {
        // Terminate the instance
        var ec2Client = new AmazonEC2Client();
        await TerminateInstance(ec2Client, args[0]);
      }
      else
      {
        Console.WriteLine("\nCommand-line argument missing or incorrect.");
```

```
        Console.WriteLine("\nUsage: EC2TerminateInstance instance-ID");
        Console.WriteLine("  instance-ID - The EC2 instance you want to terminate.");
        return;
      }
    }

    //
    // Method to terminate an EC2 instance
    private static async Task TerminateInstance(IAmazonEC2 ec2Client, string
 instanceID)
    {
      var request = new TerminateInstancesRequest{
        InstanceIds = new List<string>() { instanceID }};
      TerminateInstancesResponse response =
        await ec2Client.TerminateInstancesAsync(new TerminateInstancesRequest{
          InstanceIds = new List<string>() { instanceID }
        });
      foreach (InstanceStateChange item in response.TerminatingInstances)
      {
        Console.WriteLine("Terminated instance: " + item.InstanceId);
        Console.WriteLine("Instance state: " + item.CurrentState.Name);
      }
    }
  }
}
```

Setelah Anda menjalankan contoh, sebaiknya masuk ke konsol Amazon EC2 untuk memverifikasi bahwa instans EC2 telah dihentikan.

Tutorial Instans Spot Amazon EC2

Tutorial ini menunjukkan cara menggunakan AWS SDK untuk .NET untuk mengelola Instans Spot Amazon EC2.

Ikhtisar

Instans Spot memungkinkan Anda meminta kapasitas Amazon EC2 yang tidak digunakan dengan harga kurang dari harga Sesuai Permintaan. Ini secara signifikan dapat menurunkan biaya EC2 Anda untuk aplikasi yang dapat terganggu.

Berikut ini adalah ringkasan tingkat tinggi tentang bagaimana Instans Spot diminta dan digunakan.

1. Buat permintaan Instans Spot, tentukan harga maksimum yang bersedia Anda bayar.

2. Ketika permintaan terpenuhi, jalankan instance seperti yang Anda lakukan pada instans Amazon EC2 lainnya.
3. Jalankan instance selama yang Anda inginkan dan kemudian hentikan, kecuali jika Harga Spot berubah sedemikian rupa sehingga instance dihentikan untuk Anda.
4. Bersihkan permintaan Instans Spot saat Anda tidak lagi membutuhkannya sehingga Instans Spot tidak lagi dibuat.

Ini telah menjadi ikhtisar tingkat yang sangat tinggi dari Instans Spot. Untuk mendapatkan pemahaman yang lebih baik tentang Instans Spot, lihat Instans Spot di Panduan Pengguna Amazon EC2.

Tentang tutorial ini

Ketika Anda mengikuti tutorial ini, Anda menggunakan AWS SDK untuk .NET untuk melakukan hal berikut:

- Membuat permintaan Instans Spot
- Tentukan kapan permintaan Instans Spot telah terpenuhi
- Batalkan permintaan Instans Spot
- Mengakhiri instance terkait

Bagian berikut menyediakan cuplikan dan informasi lain untuk contoh ini. Kode lengkap untuk contoh ditampilkan setelah cuplikan, dan dapat dibangun dan dijalankan apa adanya.

Topik

- Prasyarat
- Kumpulkan apa yang Anda butuhkan
- Membuat permintaan Instans Spot
- Tentukan status permintaan Instans Spot Anda
- Bersihkan permintaan Instans Spot Anda
- Bersihkan Instans Spot Anda
- Kode lengkap
- Pertimbangan tambahan

Prasyarat

Untuk informasi tentang APIs dan prasyarat, lihat bagian induk (). Bekerja dengan Amazon EC2

Kumpulkan apa yang Anda butuhkan

Untuk membuat permintaan Instans Spot, Anda memerlukan beberapa hal.

- Jumlah instance dan jenis instance-nya. Ada beberapa jenis contoh untuk dipilih. Untuk informasi selengkapnya, lihat jenis instans Amazon EC2 di Panduan Pengguna Amazon EC2. Lihat juga Detail Jenis Instance dan Penjelajah Jenis Instance.

  Nomor default untuk tutorial ini adalah 1.

- Amazon Machine Image (AMI) yang akan digunakan untuk membuat instance. Untuk selengkapnya AMIs, lihat Gambar Mesin Amazon (AMIs) di Panduan Pengguna Amazon EC2. Secara khusus, lihat Menemukan AMI dan Dibagikan AMIs.

- Harga maksimum yang bersedia Anda bayar per jam instans. Anda dapat melihat harga untuk semua jenis instans (untuk Instans Sesuai Permintaan dan Instans Spot) di halaman harga Amazon EC2. Harga default untuk tutorial ini dijelaskan nanti.

- Jika Anda ingin terhubung dari jarak jauh ke sebuah instans, grup keamanan dengan konfigurasi dan sumber daya yang sesuai. Ini dijelaskan dalam Bekerja dengan grup keamanan di Amazon EC2 dan informasi tentang mengumpulkan apa yang Anda butuhkan dan menghubungkan ke instance diMeluncurkan instans Amazon EC2. Untuk mempermudah, tutorial ini menggunakan grup keamanan bernama default yang dimiliki semua AWS akun yang lebih baru.

Ada banyak cara untuk mendekati permintaan Instans Spot. Berikut ini adalah strategi umum:

- Buat permintaan yang pasti biayanya kurang dari harga sesuai permintaan.
- Buat permintaan berdasarkan nilai perhitungan yang dihasilkan.
- Buat permintaan untuk memperoleh kapasitas komputasi secepat mungkin.

Penjelasan berikut mengacu pada riwayat harga Instans Spot di Panduan Pengguna Amazon EC2.

Mengurangi biaya di bawah On-Demand

Anda memiliki pekerjaan pemrosesan batch yang akan memakan waktu beberapa jam atau hari untuk dijalankan. Namun, Anda fleksibel sehubungan dengan kapan dimulai dan berakhir. Anda ingin melihat apakah Anda dapat menyelesaikannya dengan harga kurang dari biaya Instans Sesuai Permintaan.

Anda memeriksa riwayat Harga Spot untuk jenis instans dengan menggunakan konsol Amazon EC2 atau Amazon EC2 API. Setelah menganalisis riwayat harga untuk jenis instans yang diinginkan di Availability Zone tertentu, Anda memiliki dua pendekatan alternatif untuk permintaan Anda:

- Tentukan permintaan di ujung atas kisaran Harga Spot, yang masih di bawah harga Sesuai Permintaan, mengantisipasi bahwa permintaan Instans Spot satu kali Anda kemungkinan besar akan terpenuhi dan berjalan untuk waktu komputasi yang cukup berturut-turut untuk menyelesaikan pekerjaan.
- Tentukan permintaan di ujung bawah kisaran harga, dan rencanakan untuk menggabungkan banyak instance yang diluncurkan dari waktu ke waktu melalui permintaan persisten. Contoh akan berjalan cukup lama, secara agregat, untuk menyelesaikan pekerjaan dengan biaya total yang lebih rendah.

Bayar tidak lebih dari nilai hasilnya

Anda memiliki pekerjaan pemrosesan data untuk dijalankan. Anda memahami nilai hasil pekerjaan dengan cukup baik untuk mengetahui berapa nilainya dalam hal biaya komputasi.

Setelah menganalisis riwayat Harga Spot untuk jenis instans Anda, Anda memilih harga di mana biaya waktu komputasi tidak lebih dari nilai hasil pekerjaan. Anda membuat permintaan persisten dan mengizinkannya berjalan sebentar-sebentar karena Harga Spot berfluktuasi pada atau di bawah permintaan Anda.

Memperoleh kapasitas komputasi dengan cepat

Anda memiliki kebutuhan jangka pendek yang tidak terduga untuk kapasitas tambahan yang tidak tersedia melalui Instans Sesuai Permintaan. Setelah menganalisis riwayat Harga Spot untuk jenis instans Anda, Anda memilih harga di atas harga historis tertinggi untuk meningkatkan kemungkinan permintaan Anda akan terpenuhi dengan cepat dan terus menghitung hingga selesai.

Setelah Anda mengumpulkan apa yang Anda butuhkan dan memilih strategi, Anda siap untuk meminta Instans Spot. Untuk tutorial ini, harga spot-instance maksimum default ditetapkan sama

dengan harga On-Demand (yaitu $0,003 untuk tutorial ini). Menetapkan harga dengan cara ini memaksimalkan kemungkinan permintaan akan dipenuhi.

Membuat permintaan Instans Spot

Cuplikan berikut menunjukkan cara membuat permintaan Instance Spot dengan elemen yang Anda kumpulkan sebelumnya.

Contoh di akhir topik ini menunjukkan cuplikan ini digunakan.

```
//
// Method to create a Spot Instance request
private static async Task<SpotInstanceRequest> CreateSpotInstanceRequest(
  IAmazonEC2 ec2Client, string amiId, string securityGroupName,
  InstanceType instanceType, string spotPrice, int instanceCount)
{
  var launchSpecification = new LaunchSpecification{
    ImageId = amiId,
    InstanceType = instanceType
  };
  launchSpecification.SecurityGroups.Add(securityGroupName);
  var request = new RequestSpotInstancesRequest{
    SpotPrice = spotPrice,
    InstanceCount = instanceCount,
    LaunchSpecification = launchSpecification
  };

  RequestSpotInstancesResponse result =
    await ec2Client.RequestSpotInstancesAsync(request);
  return result.SpotInstanceRequests[0];
}
```

Nilai penting yang dikembalikan dari metode ini adalah ID permintaan Instance Spot, yang terkandung dalam `SpotInstanceRequestId` anggota SpotInstanceRequestobjek yang dikembalikan.

> **Note**
>
> Anda akan dikenakan biaya untuk Instans Spot apa pun yang diluncurkan. Untuk menghindari biaya yang tidak perlu, pastikan untuk membatalkan permintaan apa pun dan menghentikan instance apa pun.

Tentukan status permintaan Instans Spot Anda

Cuplikan berikut menunjukkan cara mendapatkan informasi tentang permintaan Instans Spot Anda. Anda dapat menggunakan informasi tersebut untuk membuat keputusan tertentu dalam kode Anda, seperti apakah akan terus menunggu permintaan Instans Spot dipenuhi.

Contoh di akhir topik ini menunjukkan cuplikan ini digunakan.

```
//
// Method to get information about a Spot Instance request, including the status,
// instance ID, etc.
// It gets the information for a specific request (as opposed to all requests).
private static async Task<SpotInstanceRequest> GetSpotInstanceRequestInfo(
  IAmazonEC2 ec2Client, string requestId)
{
  var describeRequest = new DescribeSpotInstanceRequestsRequest();
  describeRequest.SpotInstanceRequestIds.Add(requestId);

  DescribeSpotInstanceRequestsResponse describeResponse =
    await ec2Client.DescribeSpotInstanceRequestsAsync(describeRequest);
  return describeResponse.SpotInstanceRequests[0];
}
```

Metode ini mengembalikan informasi tentang permintaan Instans Spot seperti ID instance, statusnya, dan kode status. Untuk informasi selengkapnya tentang kode status permintaan Instans Spot, lihat Status permintaan Spot di Panduan Pengguna Amazon EC2.

Bersihkan permintaan Instans Spot Anda

Bila Anda tidak perlu lagi meminta Instans Spot, penting untuk membatalkan permintaan yang belum selesai untuk mencegah permintaan tersebut terpenuhi kembali. Cuplikan berikut menunjukkan cara membatalkan permintaan Instans Spot.

Contoh di akhir topik ini menunjukkan cuplikan ini digunakan.

```
//
// Method to cancel a Spot Instance request
private static async Task CancelSpotInstanceRequest(
  IAmazonEC2 ec2Client, string requestId)
{
  var cancelRequest = new CancelSpotInstanceRequestsRequest();
```

```
      cancelRequest.SpotInstanceRequestIds.Add(requestId);

      await ec2Client.CancelSpotInstanceRequestsAsync(cancelRequest);
    }
```

Bersihkan Instans Spot Anda

Untuk menghindari biaya yang tidak perlu, penting bagi Anda untuk menghentikan instans apa pun yang dimulai dari permintaan Instans Spot; hanya membatalkan permintaan Instans Spot tidak akan menghentikan instans Anda, yang berarti Anda akan terus dikenakan biaya untuk itu. Cuplikan berikut menunjukkan cara menghentikan instance setelah mendapatkan pengenal instans untuk Instance Spot yang aktif.

Contoh di akhir topik ini menunjukkan cuplikan ini digunakan.

```
    //
    // Method to terminate a Spot Instance
    private static async Task TerminateSpotInstance(
      IAmazonEC2 ec2Client, string requestId)
    {
      var describeRequest = new DescribeSpotInstanceRequestsRequest();
      describeRequest.SpotInstanceRequestIds.Add(requestId);

      // Retrieve the Spot Instance request to check for running instances.
      DescribeSpotInstanceRequestsResponse describeResponse =
        await ec2Client.DescribeSpotInstanceRequestsAsync(describeRequest);

      // If there are any running instances, terminate them
      if(   (describeResponse.SpotInstanceRequests[0].Status.Code
              == "request-canceled-and-instance-running")
         || (describeResponse.SpotInstanceRequests[0].State ==
SpotInstanceState.Active))
      {
        TerminateInstancesResponse response =
          await ec2Client.TerminateInstancesAsync(new TerminateInstancesRequest{
            InstanceIds = new List<string>(){
              describeResponse.SpotInstanceRequests[0].InstanceId } });
        foreach (InstanceStateChange item in response.TerminatingInstances)
        {
          Console.WriteLine($"\n  Terminated instance: {item.InstanceId}");
          Console.WriteLine($"  Instance state: {item.CurrentState.Name}\n");
        }
      }
```

```
    }
```

Kode lengkap

Contoh kode berikut memanggil metode yang dijelaskan sebelumnya untuk membuat dan membatalkan permintaan Instans Spot dan mengakhiri Instance Spot.

Referensi SDK

NuGet paket:

- AWSSDK.EC2

Elemen pemrograman:

- Namespace Amazon.EC2

  EC2Klien Kelas Amazon

  Kelas InstanceType

- Namespace Amazon.EC2.Model

  Kelas CancelSpotInstanceRequestsRequest

  Kelas DescribeSpotInstanceRequestsRequest

  Kelas DescribeSpotInstanceRequestsResponse

  Kelas InstanceStateChange

  Kelas LaunchSpecification

  Kelas RequestSpotInstancesRequest

  Kelas RequestSpotInstancesResponse

  Kelas SpotInstanceRequest

  Kelas TerminateInstancesRequest

  Kelas TerminateInstancesResponse

Kodenya

```
using System;
using System.Threading;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2SpotInstanceRequests
{
  class Program
  {
    static async Task Main(string[] args)
    {
      // Some default values.
      // These could be made into command-line arguments instead.
      var instanceType = InstanceType.T1Micro;
      string securityGroupName = "default";
      string spotPrice = "0.003";
      int instanceCount = 1;

      // Parse the command line arguments
      if((args.Length != 1) || (!args[0].StartsWith("ami-")))
      {
        Console.WriteLine("\nUsage: EC2SpotInstanceRequests ami");
        Console.WriteLine("  ami: the Amazon Machine Image to use for the Spot
 Instances.");
        return;
      }

      // Create the Amazon EC2 client.
      var ec2Client = new AmazonEC2Client();

      // Create the Spot Instance request and record its ID
      Console.WriteLine("\nCreating spot instance request...");
      var req = await CreateSpotInstanceRequest(
        ec2Client, args[0], securityGroupName, instanceType, spotPrice, instanceCount);
      string requestId = req.SpotInstanceRequestId;

      // Wait for an EC2 Spot Instance to become active
      Console.WriteLine(
        $"Waiting for Spot Instance request with ID {requestId} to become active...");
      int wait = 1;
```

```
    var start = DateTime.Now;
    while(true)
    {
      Console.Write(".");

      // Get and check the status to see if the request has been fulfilled.
      var requestInfo = await GetSpotInstanceRequestInfo(ec2Client, requestId);
      if(requestInfo.Status.Code == "fulfilled")
      {
        Console.WriteLine($"\nSpot Instance request {requestId} " +
          $"has been fulfilled by instance {requestInfo.InstanceId}.\n");
        break;
      }

      // Wait a bit and try again, longer each time (1, 2, 4, ...)
      Thread.Sleep(wait);
      wait = wait * 2;
    }

    // Show the user how long it took to fulfill the Spot Instance request.
    TimeSpan span = DateTime.Now.Subtract(start);
    Console.WriteLine($"That took {span.TotalMilliseconds} milliseconds");

    // Perform actions here as needed.
    // For this example, simply wait for the user to hit a key.
    // That gives them a chance to look at the EC2 console to see
    // the running instance if they want to.
    Console.WriteLine("Press any key to start the cleanup...");
    Console.ReadKey(true);

    // Cancel the request.
    // Do this first to make sure that the request can't be re-fulfilled
    // once the Spot Instance has been terminated.
    Console.WriteLine("Canceling Spot Instance request...");
    await CancelSpotInstanceRequest(ec2Client, requestId);

    // Terminate the Spot Instance that's running.
    Console.WriteLine("Terminating the running Spot Instance...");
    await TerminateSpotInstance(ec2Client, requestId);

    Console.WriteLine("Done. Press any key to exit...");
    Console.ReadKey(true);
  }
```

```
//
// Method to create a Spot Instance request
private static async Task<SpotInstanceRequest> CreateSpotInstanceRequest(
  IAmazonEC2 ec2Client, string amiId, string securityGroupName,
  InstanceType instanceType, string spotPrice, int instanceCount)
{
  var launchSpecification = new LaunchSpecification{
    ImageId = amiId,
    InstanceType = instanceType
  };
  launchSpecification.SecurityGroups.Add(securityGroupName);
  var request = new RequestSpotInstancesRequest{
    SpotPrice = spotPrice,
    InstanceCount = instanceCount,
    LaunchSpecification = launchSpecification
  };

  RequestSpotInstancesResponse result =
    await ec2Client.RequestSpotInstancesAsync(request);
  return result.SpotInstanceRequests[0];
}


//
// Method to get information about a Spot Instance request, including the status,
// instance ID, etc.
// It gets the information for a specific request (as opposed to all requests).
private static async Task<SpotInstanceRequest> GetSpotInstanceRequestInfo(
  IAmazonEC2 ec2Client, string requestId)
{
  var describeRequest = new DescribeSpotInstanceRequestsRequest();
  describeRequest.SpotInstanceRequestIds.Add(requestId);

  DescribeSpotInstanceRequestsResponse describeResponse =
    await ec2Client.DescribeSpotInstanceRequestsAsync(describeRequest);
  return describeResponse.SpotInstanceRequests[0];
}


//
// Method to cancel a Spot Instance request
private static async Task CancelSpotInstanceRequest(
  IAmazonEC2 ec2Client, string requestId)
```

```
    {
      var cancelRequest = new CancelSpotInstanceRequestsRequest();
      cancelRequest.SpotInstanceRequestIds.Add(requestId);

      await ec2Client.CancelSpotInstanceRequestsAsync(cancelRequest);
    }


    //
    // Method to terminate a Spot Instance
    private static async Task TerminateSpotInstance(
      IAmazonEC2 ec2Client, string requestId)
    {
      var describeRequest = new DescribeSpotInstanceRequestsRequest();
      describeRequest.SpotInstanceRequestIds.Add(requestId);

      // Retrieve the Spot Instance request to check for running instances.
      DescribeSpotInstanceRequestsResponse describeResponse =
        await ec2Client.DescribeSpotInstanceRequestsAsync(describeRequest);

      // If there are any running instances, terminate them
      if(   (describeResponse.SpotInstanceRequests[0].Status.Code
              == "request-canceled-and-instance-running")
         || (describeResponse.SpotInstanceRequests[0].State ==
 SpotInstanceState.Active))
      {
        TerminateInstancesResponse response =
          await ec2Client.TerminateInstancesAsync(new TerminateInstancesRequest{
            InstanceIds = new List<string>(){
              describeResponse.SpotInstanceRequests[0].InstanceId } });
        foreach (InstanceStateChange item in response.TerminatingInstances)
        {
          Console.WriteLine($"\n  Terminated instance: {item.InstanceId}");
          Console.WriteLine($"  Instance state: {item.CurrentState.Name}\n");
        }
      }
    }
  }
}
```

Pertimbangan tambahan

- Setelah Anda menjalankan tutorial, sebaiknya masuk ke konsol Amazon EC2 untuk memverifikasi bahwa permintaan Instans Spot telah dibatalkan dan Instans Spot telah dihentikan.

Mengakses AWS Identity and Access Management (IAM) dengan AWS SDK untuk .NET

AWS SDK untuk .NET Dukungan AWS Identity and Access Management, yang merupakan layanan web yang memungkinkan AWS pelanggan untuk mengelola pengguna dan izin pengguna di AWS.

Pengguna AWS Identity and Access Management (IAM) adalah entitas yang Anda buat. AWS Entitas mewakili orang atau aplikasi yang berinteraksi dengannya AWS. Untuk informasi selengkapnya tentang pengguna IAM, lihat Pengguna IAM dan Batas IAM dan STS di Panduan Pengguna IAM.

Anda memberikan izin kepada pengguna dengan membuat kebijakan IAM. Kebijakan tersebut berisi dokumen kebijakan yang mencantumkan tindakan yang dapat dilakukan pengguna dan sumber daya yang dapat memengaruhi tindakan tersebut. Untuk informasi selengkapnya tentang kebijakan IAM, lihat Kebijakan dan Izin di Panduan Pengguna IAM.

> **Warning**
>
> Untuk menghindari risiko keamanan, jangan gunakan pengguna IAM 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.

APIs

AWS SDK untuk .NET Menyediakan APIs untuk klien IAM. Ini APIs memungkinkan Anda untuk bekerja dengan fitur IAM seperti pengguna, peran, dan kunci akses.

Bagian ini berisi sejumlah kecil contoh yang menunjukkan pola yang dapat Anda ikuti saat bekerja dengan ini APIs. Untuk melihat set lengkap APIs, lihat Referensi AWS SDK untuk .NET API (dan gulir ke “Amazon. IdentityManagement“).

Bagian ini juga berisi contoh yang menunjukkan kepada Anda cara melampirkan peran IAM ke instans Amazon EC2 untuk mempermudah pengelolaan kredensyal.

IAM APIs disediakan oleh. AWSSDK IdentityManagement NuGetpaket.

Prasyarat

Sebelum Anda mulai, pastikan Anda telah mengatur lingkungan Anda dan mengkonfigurasi proyek Anda. Juga tinjau informasi diMenggunakan SDK .

Topik

Topik

- Membuat kebijakan terkelola IAM dari JSON
- Menampilkan dokumen kebijakan kebijakan yang dikelola IAM
- Memberikan akses dengan menggunakan peran IAM

Membuat kebijakan terkelola IAM dari JSON

Contoh ini menunjukkan cara menggunakan AWS SDK untuk .NET untuk membuat kebijakan terkelola IAM dari dokumen kebijakan yang diberikan di JSON. Aplikasi membuat objek klien IAM, membaca dokumen kebijakan dari file, dan kemudian membuat kebijakan.

> **Note**
>
> Untuk contoh dokumen kebijakan di JSON, lihat pertimbangan tambahan di akhir topik ini.

Bagian berikut menyediakan cuplikan dari contoh ini. Kode lengkap untuk contoh ditampilkan setelah itu, dan dapat dibangun dan dijalankan apa adanya.

Topik

- Buat kebijakan
- Kode lengkap
- Pertimbangan tambahan

Buat kebijakan

Cuplikan berikut membuat kebijakan terkelola IAM dengan nama dan dokumen kebijakan yang diberikan.

Contoh di akhir topik ini menunjukkan cuplikan ini digunakan.

```
    //
    // Method to create an IAM policy from a JSON file
    private static async Task<CreatePolicyResponse> CreateManagedPolicy(
      IAmazonIdentityManagementService iamClient, string policyName, string
 jsonFilename)
    {
      return await iamClient.CreatePolicyAsync(new CreatePolicyRequest{
        PolicyName = policyName,
        PolicyDocument = File.ReadAllText(jsonFilename)});
    }
```

Kode lengkap

Bagian ini menunjukkan referensi yang relevan dan kode lengkap untuk contoh ini.

Referensi SDK

NuGet paket:

- AWSSDK.IdentityManagement

Elemen pemrograman:

- Namespace Amazon. IdentityManagement

  Kelas AmazonIdentityManagementServiceClient
- Namespace Amazon. IdentityManagement.Model

  Kelas CreatePolicyRequest

  Kelas CreatePolicyResponse

Kodenya

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Amazon.IdentityManagement;
using Amazon.IdentityManagement.Model;
```

```
namespace IamCreatePolicyFromJson
{
  // = = =
```

```
    private static async Task<CreatePolicyResponse> CreateManagedPolicy(
      IAmazonIdentityManagementService iamClient, string policyName, string
jsonFilename)
    {
      return await iamClient.CreatePolicyAsync(new CreatePolicyRequest{
        PolicyName = policyName,
        PolicyDocument = File.ReadAllText(jsonFilename)});
    }


    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: IamCreatePolicyFromJson -p <policy-name> -j <json-filename>" +
        "\n  -p, --policy-name: The name you want the new policy to have." +
        "\n  -j, --json-filename: The name of the JSON file with the policy
document.");
    }
 }


 // = = =
```

```
    {
      var parsedArgs = new Dictionary<string,string>();
      int i = 0, n = 0;
      while(i < args.Length)
      {
        // If the first argument in this iteration starts with a dash it's an option.
        if(args[i].StartsWith("-"))
        {
          var key = args[i++];
          var value = key;

          // Check to see if there's a value that goes with this option?
          if((i < args.Length) && (!args[i].StartsWith("-"))) value = args[i++];
          parsedArgs.Add(key, value);
        }

        // If the first argument in this iteration doesn't start with a dash, it's a
value
        else
        {
          parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
          n++;
        }
      }

      return parsedArgs;
    }

    //
    // Method to get an argument from the parsed command-line arguments
    //
    // Parameters:
    // - parsedArgs: The Dictionary object returned from the Parse() method (shown
above).
    // - defaultValue: The default string to return if the specified key isn't in
parsedArgs.
    // - keys: An array of keys to look for in parsedArgs.
    public static string GetArgument(
      Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
    {
      string retval = null;
      foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
      return retval ?? defaultReturn;
```

```
    }

    //
    // Method to exit the application with an error.
    public static void ErrorExit(string msg, int code=1)
    {
      Console.WriteLine("\nError");
      Console.WriteLine(msg);
      Environment.Exit(code);
    }
  }

}
```

Pertimbangan tambahan

- Berikut ini adalah contoh dokumen kebijakan yang dapat Anda salin ke dalam file JSON dan gunakan sebagai masukan untuk aplikasi ini:

  JSON

```
{
  "Version":"2012-10-17",
  "Id"  : "DotnetTutorialPolicy",
  "Statement" : [
    {
      "Sid" : "DotnetTutorialPolicyS3",
      "Effect" : "Allow",
      "Action" : [
        "s3:Get*",
        "s3:List*"
      ],
      "Resource" : "*"
    },
    {
      "Sid" : "DotnetTutorialPolicyPolly",
      "Effect": "Allow",
      "Action": [
        "polly:DescribeVoices",
        "polly:SynthesizeSpeech"
      ],
      "Resource": "*"
    }
```

```
  ]
}
```

- Anda dapat memverifikasi bahwa kebijakan telah dibuat dengan melihat di konsol IAM. Di daftar drop-down Filter policy, pilih Pelanggan dikelola. Hapus kebijakan saat Anda tidak lagi membutuhkannya.

- Untuk informasi selengkapnya tentang pembuatan kebijakan, lihat Membuat kebijakan IAM dan referensi kebijakan IAM JSON di Panduan Pengguna IAM

Menampilkan dokumen kebijakan kebijakan yang dikelola IAM

Contoh ini menunjukkan cara menggunakan AWS SDK untuk .NET untuk menampilkan dokumen kebijakan. Aplikasi membuat objek klien IAM, menemukan versi default dari kebijakan terkelola IAM yang diberikan, dan kemudian menampilkan dokumen kebijakan di JSON.

Bagian berikut menyediakan cuplikan dari contoh ini. Kode lengkap untuk contoh ditampilkan setelah itu, dan dapat dibangun dan dijalankan apa adanya.

Topik

- Temukan versi default
- Menampilkan dokumen kebijakan
- Kode lengkap

Temukan versi default

Cuplikan berikut menemukan versi default dari kebijakan IAM yang diberikan.

Contoh di akhir topik ini menunjukkan cuplikan ini digunakan.

```
//
// Method to determine the default version of an IAM policy
// Returns a string with the version
private static async Task<string> GetDefaultVersion(
  IAmazonIdentityManagementService iamClient, string policyArn)
{
  // Retrieve all the versions of this policy
```

```
      string defaultVersion = string.Empty;
      ListPolicyVersionsResponse reponseVersions =
        await iamClient.ListPolicyVersionsAsync(new ListPolicyVersionsRequest{
          PolicyArn = policyArn});

      // Find the default version
      foreach(PolicyVersion version in reponseVersions.Versions)
      {
        if(version.IsDefaultVersion)
        {
          defaultVersion = version.VersionId;
          break;
        }
      }

      return defaultVersion;
    }
```

Menampilkan dokumen kebijakan

Cuplikan berikut menampilkan dokumen kebijakan di JSON dari kebijakan IAM yang diberikan.

Contoh di akhir topik ini menunjukkan cuplikan ini digunakan.

```
    //
    // Method to retrieve and display the policy document of an IAM policy
    private static async Task ShowPolicyDocument(
      IAmazonIdentityManagementService iamClient, string policyArn, string
defaultVersion)
    {
      // Retrieve the policy document of the default version
      GetPolicyVersionResponse responsePolicy =
        await iamClient.GetPolicyVersionAsync(new GetPolicyVersionRequest{
          PolicyArn = policyArn,
          VersionId = defaultVersion});

      // Display the policy document (in JSON)
      Console.WriteLine($"Version {defaultVersion} of the policy (in JSON format):");
      Console.WriteLine(
        $"{HttpUtility.UrlDecode(responsePolicy.PolicyVersion.Document)}");
    }
```

Kode lengkap

Bagian ini menunjukkan referensi yang relevan dan kode lengkap untuk contoh ini.

Referensi SDK

NuGet paket:

- AWSSDK.IdentityManagement

Elemen pemrograman:

- Namespace Amazon. IdentityManagement

  Kelas AmazonIdentityManagementServiceClient
- Namespace Amazon. IdentityManagement.Model

  Kelas GetPolicyVersionRequest

  Kelas GetPolicyVersionResponse

  Kelas ListPolicyVersionsRequest

  Kelas ListPolicyVersionsResponse

  Kelas PolicyVersion

Kodenya

```
using System;
using System.Web;
using System.Threading.Tasks;
using Amazon.IdentityManagement;
using Amazon.IdentityManagement.Model;

namespace IamDisplayPolicyJson
{
  class Program
  {
    static async Task Main(string[] args)
    {
      // Parse the command line and show help if necessary
```

```
      if(args.Length != 1)
      {
        Console.WriteLine("\nUsage: IamDisplayPolicyJson policy-arn");
        Console.WriteLine("   policy-arn: The ARN of the policy to retrieve.");
        return;
      }
      if(!args[0].StartsWith("arn:"))
      {
        Console.WriteLine("\nCould not find policy ARN in the command-line
arguments:");
        Console.WriteLine($"{args[0]}");
        return;
      }

      // Create an IAM service client
      var iamClient = new AmazonIdentityManagementServiceClient();

      // Retrieve and display the policy document of the given policy
      string defaultVersion = await GetDefaultVersion(iamClient, args[0]);
      if(string.IsNullOrEmpty(defaultVersion))
        Console.WriteLine($"Could not find the default version for policy {args[0]}.");
      else
        await ShowPolicyDocument(iamClient, args[0], defaultVersion);
    }


    //
    // Method to determine the default version of an IAM policy
    // Returns a string with the version
    private static async Task<string> GetDefaultVersion(
      IAmazonIdentityManagementService iamClient, string policyArn)
    {
      // Retrieve all the versions of this policy
      string defaultVersion = string.Empty;
      ListPolicyVersionsResponse reponseVersions =
        await iamClient.ListPolicyVersionsAsync(new ListPolicyVersionsRequest{
          PolicyArn = policyArn});

      // Find the default version
      foreach(PolicyVersion version in reponseVersions.Versions)
      {
        if(version.IsDefaultVersion)
        {
          defaultVersion = version.VersionId;
```

```
          break;
        }
      }

      return defaultVersion;
    }


    //
    // Method to retrieve and display the policy document of an IAM policy
    private static async Task ShowPolicyDocument(
      IAmazonIdentityManagementService iamClient, string policyArn, string
 defaultVersion)
    {
      // Retrieve the policy document of the default version
      GetPolicyVersionResponse responsePolicy =
        await iamClient.GetPolicyVersionAsync(new GetPolicyVersionRequest{
          PolicyArn = policyArn,
          VersionId = defaultVersion});

      // Display the policy document (in JSON)
      Console.WriteLine($"Version {defaultVersion} of the policy (in JSON format):");
      Console.WriteLine(
        $"{HttpUtility.UrlDecode(responsePolicy.PolicyVersion.Document)}");
    }
  }
}
```

Memberikan akses dengan menggunakan peran IAM

Tutorial ini menunjukkan cara menggunakan AWS SDK untuk .NET untuk mengaktifkan peran IAM pada instans Amazon EC2.

Ikhtisar

Semua permintaan AWS harus ditandatangani secara kriptografi dengan menggunakan kredensil yang dikeluarkan oleh. AWS Oleh karena itu, Anda memerlukan strategi untuk mengelola kredensional untuk aplikasi yang berjalan di instans Amazon EC2. Anda harus mendistribusikan, menyimpan, dan memutar kredensi ini dengan aman, tetapi juga membuatnya dapat diakses oleh aplikasi.

Dengan peran IAM, Anda dapat mengelola kredensil ini secara efektif. Anda membuat peran IAM dan mengonfigurasinya dengan izin yang diperlukan aplikasi, lalu melampirkan peran itu ke instans

EC2. Untuk membaca selengkapnya tentang manfaat menggunakan peran IAM, lihat peran IAM untuk Amazon EC2 di Panduan Pengguna Amazon EC2. Lihat juga informasi tentang Peran IAM di Panduan Pengguna IAM.

Untuk aplikasi yang dibangun menggunakan AWS SDK untuk .NET, ketika aplikasi membangun objek klien untuk AWS layanan, objek mencari kredensil dari beberapa sumber potensial. Urutan penelusurannya ditampilkan diResolusi kredensi dan profil.

Jika objek klien tidak menemukan kredensil dari sumber lain, ia mengambil kredensil sementara yang memiliki izin yang sama dengan yang telah dikonfigurasi ke dalam peran IAM dan berada dalam metadata instans EC2. Kredensial ini digunakan untuk melakukan panggilan ke AWS dari objek klien.

Tentang tutorial ini

Saat Anda mengikuti tutorial ini, Anda menggunakan AWS SDK untuk .NET (dan alat lainnya) untuk meluncurkan instans Amazon EC2 dengan peran IAM terlampir, dan kemudian melihat aplikasi pada instance menggunakan izin peran IAM.

Topik

- Buat contoh aplikasi Amazon S3
- Membuat peran IAM
- Luncurkan instans EC2 dan lampirkan peran IAM
- Connect ke instans EC2
- Jalankan aplikasi sampel pada instans EC2
- Bersihkan

Buat contoh aplikasi Amazon S3

Contoh aplikasi ini mengambil objek dari Amazon S3. Untuk menjalankan aplikasi, Anda memerlukan yang berikut ini:

- Bucket Amazon S3 yang berisi file teks.
- AWS kredensi pada mesin pengembangan Anda yang memungkinkan Anda mengakses bucket.

Untuk informasi tentang membuat bucket Amazon S3 dan mengunggah objek, lihat Panduan Pengguna Layanan Penyimpanan Sederhana Amazon. Untuk informasi tentang AWS kredensil, lihat. Mengautentikasi dengan AWS SDK untuk .NET AWS

Buat proyek .NET Core dengan kode berikut. Kemudian uji aplikasi pada mesin pengembangan Anda.

> **Note**
>
> Pada mesin pengembangan Anda, .NET Core Runtime diinstal, yang memungkinkan Anda menjalankan aplikasi tanpa mempublikasikannya. Ketika Anda membuat instans EC2 nanti dalam tutorial ini, Anda dapat memilih untuk menginstal .NET Core Runtime pada instance. Ini memberi Anda pengalaman serupa dan transfer file yang lebih kecil.
> Namun, Anda juga dapat memilih untuk tidak menginstal .NET Core Runtime pada instance. Jika Anda memilih tindakan ini, Anda harus mempublikasikan aplikasi sehingga semua dependensi disertakan saat Anda mentransfernya ke instance.

Referensi SDK

NuGet paket:

- AWSSDK.S3

Elemen pemrograman:

- Namespace Amazon.S3

  Kelas Amazons3Client
- Namespace Amazon.S3.Model

  Kelas GetObjectResponse

Kodenya

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

namespace S3GetTextItem
{
```

```
 // = = =
```

```
      IAmazonS3 s3Client, string bucket, string item)
    {
        Console.WriteLine($"Retrieving {item} from bucket {bucket}.");
        return await s3Client.GetObjectAsync(bucket, item);
    }


    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: S3GetTextItem -b <bucket-name> -t <text-object> -o <output-filename>"
+
        "\n  -b, --bucket-name: The name of the S3 bucket." +
        "\n  -t, --text-object: The name of the text object in the bucket." +
        "\n  -o, --output-filename: The name of the file to write the text to.");
    }
 }


 // = = =
```

```
      int i = 0, n = 0;
      while(i < args.Length)
      {
        // If the first argument in this iteration starts with a dash it's an option.
        if(args[i].StartsWith("-"))
        {
          var key = args[i++];
          var value = key;

          // Check to see if there's a value that goes with this option?
          if((i < args.Length) && (!args[i].StartsWith("-"))) value = args[i++];
          parsedArgs.Add(key, value);
        }

        // If the first argument in this iteration doesn't start with a dash, it's a
value
        else
        {
          parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
          n++;
        }
      }

      return parsedArgs;
    }

    //
    // Method to get an argument from the parsed command-line arguments
    //
    // Parameters:
    // - parsedArgs: The Dictionary object returned from the Parse() method (shown
above).
    // - defaultValue: The default string to return if the specified key isn't in
parsedArgs.
    // - keys: An array of keys to look for in parsedArgs.
    public static string GetArgument(
      Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
    {
      string retval = null;
      foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
      return retval ?? defaultReturn;
    }
```

```
    //
    // Method to exit the application with an error.
    public static void ErrorExit(string msg, int code=1)
    {
      Console.WriteLine("\nError");
      Console.WriteLine(msg);
      Environment.Exit(code);
    }
  }

}
```

Jika mau, Anda dapat menghapus sementara kredensi yang Anda gunakan pada mesin pengembangan Anda untuk melihat bagaimana aplikasi merespons. (Tapi pastikan untuk mengembalikan kredensialnya saat Anda selesai.)

Membuat peran IAM

Buat peran IAM yang memiliki izin yang sesuai untuk mengakses Amazon S3.

1. Buka konsol IAM.
2. Di panel navigasi, pilih Peran, lalu pilih Buat peran.
3. Pilih AWS layanan, temukan dan pilih EC2, dan pilih Berikutnya: Izin.
4. Di bawah Lampirkan kebijakan izin, temukan dan pilih ReadOnlyAccessAmazonS3. Tinjau kebijakan jika Anda mau, lalu pilih Berikutnya: Tag.
5. Tambahkan tag jika Anda mau dan kemudian pilih Berikutnya: Tinjau.
6. Ketik nama dan deskripsi untuk peran tersebut, lalu pilih Buat peran. Ingat nama ini karena Anda akan membutuhkannya saat meluncurkan instans EC2 Anda.

Luncurkan instans EC2 dan lampirkan peran IAM

Luncurkan instans EC2 dengan peran IAM yang Anda buat sebelumnya. Anda dapat melakukannya dengan cara-cara berikut.

- Menggunakan konsol EC2

  Untuk meluncurkan instans menggunakan konsol EC2, lihat Meluncurkan instance menggunakan wizard instans peluncuran baru di Panduan Pengguna Amazon EC2.

Saat Anda melihat melalui halaman peluncuran, Anda setidaknya harus memperluas panel Detail lanjutan sehingga Anda dapat menentukan peran IAM yang Anda buat sebelumnya di profil instans IAM.

- Menggunakan AWS SDK untuk .NET

  Untuk informasi tentang ini, lihatMeluncurkan instans Amazon EC2, termasuk di Pertimbangan tambahan dekat akhir topik itu.

Untuk meluncurkan instans EC2 yang memiliki peran IAM terpasang, konfigurasi pengguna IAM harus menyertakan izin tertentu. Untuk informasi selengkapnya tentang izin yang diperlukan, lihat Memberikan izin kepada pengguna untuk meneruskan peran IAM ke instans di Panduan Pengguna Amazon EC2.

Connect ke instans EC2

Connect ke instans EC2 sehingga Anda dapat mentransfer aplikasi sampel ke sana dan kemudian menjalankan aplikasi. Anda akan memerlukan file yang berisi bagian pribadi dari key pair yang Anda gunakan untuk meluncurkan instance; yaitu, file PEM.

Untuk informasi tentang menghubungkan ke instans, lihat Connect ke instans Linux atau Connect ke instans Windows Anda di Panduan Pengguna Amazon EC2. Ketika Anda terhubung, lakukan sedemikian rupa sehingga Anda dapat mentransfer file dari mesin pengembangan Anda ke instans Anda.

Jika Anda menggunakan Visual Studio di Windows, Anda juga dapat terhubung ke instance dengan menggunakan Toolkit for Visual Studio. Untuk informasi selengkapnya, lihat Menyambungkan ke Instans Amazon EC2 di AWS Toolkit for Visual Studio Panduan Pengguna.

Jalankan aplikasi sampel pada instans EC2

1. Salin file aplikasi dari drive lokal Anda ke instans Anda.

   File mana yang Anda transfer tergantung pada bagaimana Anda membangun aplikasi dan apakah instans Anda telah menginstal .NET Core Runtime. Untuk informasi tentang cara mentransfer file ke instans Anda, lihat Connect ke instans Linux Anda (lihat sub-bagian yang sesuai) atau Mentransfer file ke instans Windows di Panduan Pengguna Amazon EC2.

2. Mulai aplikasi dan verifikasi bahwa itu berjalan dengan hasil yang sama seperti pada mesin pengembangan Anda.
3. Verifikasi bahwa aplikasi menggunakan kredensional yang disediakan oleh peran IAM.
    a. Buka konsol Amazon EC2.
    b. Pilih instance dan lepaskan peran IAM melalui Tindakan, Pengaturan Instans, Lampirkan/Ganti Peran IAM.
    c. Jalankan aplikasi lagi dan lihat bahwa itu mengembalikan kesalahan otorisasi.

Bersihkan

Ketika Anda selesai dengan tutorial ini, dan jika Anda tidak lagi menginginkan instans EC2 yang Anda buat, pastikan untuk menghentikan instance untuk menghindari biaya yang tidak diinginkan. Anda dapat melakukannya di konsol Amazon EC2 atau secara terprogram, seperti yang dijelaskan dalam. Mengakhiri instans Amazon EC2 Jika mau, Anda juga dapat menghapus sumber daya lain yang Anda buat untuk tutorial ini. Ini mungkin termasuk peran IAM, keypair EC2 dan file PEM, grup keamanan, dll.

Menggunakan penyimpanan Internet Amazon Simple Storage Service

AWS SDK untuk .NET Mendukung Amazon S3, yang merupakan penyimpanan untuk Internet. Ini dirancang untuk membuat komputasi skala web lebih mudah bagi pengembang.

APIs

AWS SDK untuk .NET Menyediakan APIs untuk klien Amazon S3. Ini APIs memungkinkan Anda untuk bekerja dengan sumber daya Amazon S3 seperti ember dan item. Untuk melihat set lengkap APIs untuk Amazon S3, lihat yang berikut ini:

- AWS SDK untuk .NET Referensi API (dan gulir ke “Amazon.S3").
- Dokumentasi Amazon.Extensions.S3.Encryption

Amazon S3 APIs disediakan oleh paket-paket berikut: NuGet

- AWSSDK.S3
- Amazon.Extensions.S3.Encryption

Prasyarat

Sebelum Anda mulai, pastikan Anda telah mengatur lingkungan Anda dan mengkonfigurasi proyek Anda. Juga tinjau informasi diMenggunakan SDK .

Contoh dalam dokumen ini

Topik berikut dalam dokumen ini menunjukkan cara menggunakan AWS SDK untuk .NET untuk bekerja dengan Amazon S3.

- Menggunakan kunci KMS untuk enkripsi S3

Contoh dalam dokumen lain

Tautan berikut ke Panduan Pengembang Amazon S3 memberikan contoh tambahan tentang cara menggunakan AWS SDK untuk .NET untuk bekerja dengan Amazon S3.

> **Note**
>
> Meskipun contoh-contoh ini dan pertimbangan pemrograman tambahan dibuat untuk Versi 3 dari AWS SDK untuk .NET menggunakan .NET Framework, mereka juga layak untuk versi yang lebih baru dari AWS SDK untuk .NET menggunakan .NET Core. Penyesuaian kecil dalam kode terkadang diperlukan.

Contoh pemrograman Amazon S3

- Mengelola ACLs
- Membuat Bucket
- Unggah Objek
- Unggah Multipart dengan API Tingkat Tinggi (Amazon.S3.Transfer. TransferUtility)
- Unggah Multipart dengan API Tingkat Rendah
- Daftar Objek
- Kunci Daftar
- Dapatkan Objek
- Salin Objek
- Salin Objek dengan Multipart Upload API

- Menghapus Objek
- Menghapus Beberapa Objek
- Mengembalikan Objek
- Konfigurasikan Bucket untuk Pemberitahuan
- Mengelola Siklus Hidup Objek
- Menghasilkan URL Objek Pra-ditandatangani
- Mengelola Situs Web
- Mengaktifkan Berbagi Sumber Daya Lintas Asal (CORS)

Pertimbangan pemrograman tambahan

- Menggunakan AWS SDK untuk .NET untuk Pemrograman Amazon S3
- Membuat Permintaan Menggunakan Kredensial Sementara Pengguna IAM
- Membuat Permintaan Menggunakan Kredensial Sementara Pengguna Federasi
- Menentukan Enkripsi Sisi Server
- Menentukan Enkripsi Sisi Server dengan Kunci Enkripsi yang Disediakan Pelanggan

Menggunakan AWS KMS kunci untuk enkripsi Amazon S3 di AWS SDK untuk .NET

Contoh ini menunjukkan cara menggunakan AWS Key Management Service kunci untuk mengenkripsi objek Amazon S3. Aplikasi ini membuat kunci master pelanggan (CMK) dan menggunakannya untuk membuat objek AmazonS3 EncryptionClient V2 untuk enkripsi sisi klien. Aplikasi menggunakan klien tersebut untuk membuat objek terenkripsi dari file teks tertentu di bucket Amazon S3 yang ada. Kemudian mendekripsi objek dan menampilkan isinya.

> **Warning**
>
> Kelas serupa yang `AmazonS3EncryptionClient` disebut tidak digunakan lagi dan kurang aman daripada kelas. `AmazonS3EncryptionClientV2` Untuk memigrasikan kode yang ada yang menggunakan`AmazonS3EncryptionClient`, lihatMigrasi Klien Enkripsi S3 (V1 ke V2).

Topik

- Buat bahan enkripsi

- Membuat dan mengenkripsi objek Amazon S3
- Kode lengkap
- Pertimbangan tambahan

Buat bahan enkripsi

Cuplikan berikut membuat `EncryptionMaterials` objek yang berisi ID kunci KMS.

Contoh di akhir topik ini menunjukkan cuplikan ini digunakan.

```
     // Create a customer master key (CMK) and store the result
     CreateKeyResponse createKeyResponse =
       await new AmazonKeyManagementServiceClient().CreateKeyAsync(new
CreateKeyRequest());
     var kmsEncryptionContext = new Dictionary<string, string>();
     var kmsEncryptionMaterials = new EncryptionMaterialsV2(
       createKeyResponse.KeyMetadata.KeyId, KmsType.KmsContext, kmsEncryptionContext);
```

Membuat dan mengenkripsi objek Amazon S3

Cuplikan berikut membuat `AmazonS3EncryptionClientV2` objek yang menggunakan bahan enkripsi yang dibuat sebelumnya. Kemudian menggunakan klien untuk membuat dan mengenkripsi objek Amazon S3 baru.

Contoh di akhir topik ini menunjukkan cuplikan ini digunakan.

```
   //
   // Method to create and encrypt an object in an S3 bucket
   static async Task<GetObjectResponse> CreateAndRetrieveObjectAsync(
     EncryptionMaterialsV2 materials, string bucketName,
     string fileName, string itemName)
   {
     // CryptoStorageMode.ObjectMetadata is required for KMS EncryptionMaterials
     var config = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy)
     {
       StorageMode = CryptoStorageMode.ObjectMetadata
     };
     var s3EncClient = new AmazonS3EncryptionClientV2(config, materials);

     // Create, encrypt, and put the object
     await s3EncClient.PutObjectAsync(new PutObjectRequest
     {
```

```
        BucketName = bucketName,
        Key = itemName,
        ContentBody = File.ReadAllText(fileName)
      });

      // Get, decrypt, and return the object
      return await s3EncClient.GetObjectAsync(new GetObjectRequest
      {
        BucketName = bucketName,
        Key = itemName
      });
    }
```

Kode lengkap

Bagian ini menunjukkan referensi yang relevan dan kode lengkap untuk contoh ini.

Referensi SDK

NuGet paket:

- Amazon.Extensions.S3.Encryption

Elemen pemrograman:

- Namespace Amazon.Extensions.S3.Encryption

  Kelas EncryptionClientAmazonS3 V2

  Kelas CryptoConfigurationAmazonS3 V2

  Kelas CryptoStorageMode

  Kelas EncryptionMaterialsV2
- Namespace Amazon.Extensions.S3.Encryption.Primitives

  Kelas KmsType
- Namespace Amazon.S3.Model

  Kelas GetObjectRequest

  Kelas GetObjectResponse

Kelas PutObjectRequest

- Namespace Amazon. KeyManagementService

  Kelas AmazonKeyManagementServiceClient

- Namespace Amazon. KeyManagementService.Model

  Kelas CreateKeyRequest

  Kelas CreateKeyResponse

Kodenya

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;
using Amazon.S3.Model;
using Amazon.KeyManagementService;
using Amazon.KeyManagementService.Model;

namespace KmsS3Encryption
{
  // = = =
```

```
      // Get the application arguments from the parsed list
      string bucketName =
        CommandLine.GetArgument(parsedArgs, null, "-b", "--bucket-name");
      string fileName =
        CommandLine.GetArgument(parsedArgs, null, "-f", "--file-name");
      string itemName =
        CommandLine.GetArgument(parsedArgs, null, "-i", "--item-name");
      if(string.IsNullOrEmpty(bucketName) || (string.IsNullOrEmpty(fileName)))
        CommandLine.ErrorExit(
          "\nOne or more of the required arguments is missing or incorrect." +
          "\nRun the command with no arguments to see help.");
      if(!File.Exists(fileName))
        CommandLine.ErrorExit($"\nThe given file {fileName} doesn't exist.");
      if(string.IsNullOrEmpty(itemName))
        itemName = Path.GetFileName(fileName);

      // Create a customer master key (CMK) and store the result
      CreateKeyResponse createKeyResponse =
        await new AmazonKeyManagementServiceClient().CreateKeyAsync(new
CreateKeyRequest());
      var kmsEncryptionContext = new Dictionary<string, string>();
      var kmsEncryptionMaterials = new EncryptionMaterialsV2(
        createKeyResponse.KeyMetadata.KeyId, KmsType.KmsContext, kmsEncryptionContext);

      // Create the object in the bucket, then display the content of the object
      var putObjectResponse =
        await CreateAndRetrieveObjectAsync(kmsEncryptionMaterials, bucketName,
fileName, itemName);
      Stream stream = putObjectResponse.ResponseStream;
      StreamReader reader = new StreamReader(stream);
      Console.WriteLine(reader.ReadToEnd());
    }


    //
    // Method to create and encrypt an object in an S3 bucket
    static async Task<GetObjectResponse> CreateAndRetrieveObjectAsync(
      EncryptionMaterialsV2 materials, string bucketName,
      string fileName, string itemName)
    {
      // CryptoStorageMode.ObjectMetadata is required for KMS EncryptionMaterials
      var config = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy)
      {
        StorageMode = CryptoStorageMode.ObjectMetadata
```

```
      };
      var s3EncClient = new AmazonS3EncryptionClientV2(config, materials);

      // Create, encrypt, and put the object
      await s3EncClient.PutObjectAsync(new PutObjectRequest
      {
        BucketName = bucketName,
        Key = itemName,
        ContentBody = File.ReadAllText(fileName)
      });

      // Get, decrypt, and return the object
      return await s3EncClient.GetObjectAsync(new GetObjectRequest
      {
        BucketName = bucketName,
        Key = itemName
      });
    }


    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: KmsS3Encryption -b <bucket-name> -f <file-name> [-i <item-name>]" +
        "\n  -b, --bucket-name: The name of an existing S3 bucket." +
        "\n  -f, --file-name: The name of a text file with content to encrypt and store
in S3." +
        "\n  -i, --item-name: The name you want to use for the item." +
        "\n      If item-name isn't given, file-name will be used.");
    }

  }

  // = = =
```

```
    //
    // Parameters:
    // - args: The command-line arguments passed into the application by the system.
    //
    // Returns:
    // A Dictionary with string Keys and Values.
    //
    // If a key is found without a matching value, Dictionary.Value is set to the key
    //  (including the dashes).
    // If a value is found without a matching key, Dictionary.Key is set to "--NoKeyN",
    //  where "N" represents sequential numbers.
    public static Dictionary<string,string> Parse(string[] args)
    {
      var parsedArgs = new Dictionary<string,string>();
      int i = 0, n = 0;
      while(i < args.Length)
      {
        // If the first argument in this iteration starts with a dash it's an option.
        if(args[i].StartsWith("-"))
        {
          var key = args[i++];
          var value = key;

          // Check to see if there's a value that goes with this option?
          if((i < args.Length) && (!args[i].StartsWith("-"))) value = args[i++];
          parsedArgs.Add(key, value);
        }

        // If the first argument in this iteration doesn't start with a dash, it's a
value
        else
        {
          parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
          n++;
        }
      }

      return parsedArgs;
    }

    //
    // Method to get an argument from the parsed command-line arguments
    //
    // Parameters:
```

```
    // - parsedArgs: The Dictionary object returned from the Parse() method (shown
 above).
    // - defaultValue: The default string to return if the specified key isn't in
 parsedArgs.
    // - keys: An array of keys to look for in parsedArgs.
    public static string GetArgument(
      Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
    {
      string retval = null;
      foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
      return retval ?? defaultReturn;
    }

    //
    // Method to exit the application with an error.
    public static void ErrorExit(string msg, int code=1)
    {
      Console.WriteLine("\nError");
      Console.WriteLine(msg);
      Environment.Exit(code);
    }
  }

}
```

Pertimbangan tambahan

- Anda dapat memeriksa hasil dari contoh ini. Untuk melakukannya, buka konsol Amazon S3 dan buka ember yang Anda berikan ke aplikasi. Kemudian temukan objek baru, unduh, dan buka di editor teks.

- Kelas AmazonS3 EncryptionClient V2 mengimplementasikan antarmuka yang sama dengan kelas standar. `AmazonS3Client` Ini membuatnya lebih mudah untuk mem-port kode Anda ke `AmazonS3EncryptionClientV2` kelas sehingga enkripsi dan dekripsi terjadi secara otomatis dan transparan di klien.

- Salah satu keuntungan menggunakan AWS KMS kunci sebagai kunci utama Anda adalah Anda tidak perlu menyimpan dan mengelola kunci master Anda sendiri; ini dilakukan oleh AWS. Keuntungan kedua adalah bahwa `AmazonS3EncryptionClientV2` kelas interoperable dengan

`AmazonS3EncryptionClientV2` kelas. AWS SDK untuk .NET AWS SDK untuk Java Ini berarti Anda dapat mengenkripsi dengan AWS SDK untuk Java dan mendekripsi dengan AWS SDK untuk .NET, dan sebaliknya.

> **Note**
>
> `AmazonS3EncryptionClientV2`Kelas AWS SDK untuk .NET mendukung kunci master KMS hanya ketika dijalankan dalam mode metadata. Mode file instruksi dari `AmazonS3EncryptionClientV2` kelas tidak AWS SDK untuk .NET kompatibel dengan `AmazonS3EncryptionClientV2` kelas file. AWS SDK untuk Java

- Untuk informasi selengkapnya tentang enkripsi sisi klien dengan `AmazonS3EncryptionClientV2` kelas, dan cara kerja enkripsi amplop, lihat Enkripsi Data Sisi Klien dengan dan Amazon AWS SDK untuk .NET S3.

Mengirim Pemberitahuan Dari Cloud Menggunakan Amazon Simple Notification Service

> **Note**
>
> Informasi dalam topik ini khusus untuk proyek berdasarkan .NET Framework dan AWS SDK untuk .NET versi 3.3 dan sebelumnya.

Ini AWS SDK untuk .NET mendukung Amazon Simple Notification Service (Amazon SNS), yang merupakan layanan web yang memungkinkan aplikasi, pengguna akhir, dan perangkat untuk langsung mengirim notifikasi dari cloud. Untuk informasi selengkapnya, lihat Amazon SNS.

Daftar Topik Amazon SNS Anda

Contoh berikut menunjukkan cara mencantumkan topik Amazon SNS Anda, langganan untuk setiap topik, dan atribut untuk setiap topik. Contoh ini menggunakan default AmazonSimpleNotificationServiceClient.

```
// using Amazon.SimpleNotificationService;
// using Amazon.SimpleNotificationService.Model;
```

```
var client = new AmazonSimpleNotificationServiceClient();
var request = new ListTopicsRequest();
var response = new ListTopicsResponse();

do
{
  response = client.ListTopics(request);

  foreach (var topic in response.Topics)
  {
    Console.WriteLine("Topic: {0}", topic.TopicArn);

    var subs = client.ListSubscriptionsByTopic(
      new ListSubscriptionsByTopicRequest
      {
        TopicArn = topic.TopicArn
      });

    var ss = subs.Subscriptions;

    if (ss.Any())
    {
      Console.WriteLine("  Subscriptions:");

      foreach (var sub in ss)
      {
        Console.WriteLine("    {0}", sub.SubscriptionArn);
      }
    }

    var attrs = client.GetTopicAttributes(
      new GetTopicAttributesRequest
      {
        TopicArn = topic.TopicArn
      }).Attributes;

    if (attrs.Any())
    {
      Console.WriteLine("  Attributes:");

      foreach (var attr in attrs)
      {
        Console.WriteLine("    {0} = {1}", attr.Key, attr.Value);
      }
```

```
    }

    Console.WriteLine();
  }

  request.NextToken = response.NextToken;

} while (!string.IsNullOrEmpty(response.NextToken));
```

Mengirim Pesan ke Topik Amazon SNS

Contoh berikut menunjukkan cara mengirim pesan ke topik Amazon SNS. Contohnya mengambil satu argumen, ARN dari topik Amazon SNS.

```
using System;
using System.Linq;
using System.Threading.Tasks;

using Amazon;
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;

namespace SnsSendMessage
{
    class Program
    {
        static void Main(string[] args)
        {
            /* Topic ARNs must be in the correct format:
             *   arn:aws:sns:REGION:ACCOUNT_ID:NAME
             *
             *  where:
             *  REGION     is the region in which the topic is created, such as us-
west-2
             *  ACCOUNT_ID is your (typically) 12-character account ID
             *  NAME       is the name of the topic
             */
            string topicArn = args[0];
            string message = "Hello at " + DateTime.Now.ToShortTimeString();

            var client = new AmazonSimpleNotificationServiceClient(region:
 Amazon.RegionEndpoint.USWest2);
```

```
            var request = new PublishRequest
            {
                Message = message,
                TopicArn = topicArn
            };

            try
            {
                var response = client.Publish(request);

                Console.WriteLine("Message sent to topic:");
                Console.WriteLine(message);
            }
            catch (Exception ex)
            {
                Console.WriteLine("Caught exception publishing request:");
                Console.WriteLine(ex.Message);
            }
        }
    }
}
```

Lihat contoh lengkapnya, termasuk informasi tentang cara membuat dan menjalankan contoh dari baris perintah, pada GitHub.

Mengirim Pesan SMS ke Nomor Telepon

Contoh berikut menunjukkan cara mengirim pesan SMS ke nomor telepon. Contohnya mengambil satu argumen, nomor telepon, yang harus dalam salah satu dari dua format yang dijelaskan dalam komentar.

```
using System;
using System.Linq;
using System.Threading.Tasks;
using Amazon;
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;

namespace SnsPublish
{
    class Program
    {
        static void Main(string[] args)
```

```
        {
            // US phone numbers must be in the correct format:
            // +1 (nnn) nnn-nnnn OR +1nnnnnnnnnn
            string number = args[0];
            string message = "Hello at " + DateTime.Now.ToShortTimeString();

            var client = new AmazonSimpleNotificationServiceClient(region:
 Amazon.RegionEndpoint.USWest2);
            var request = new PublishRequest
            {
                Message = message,
                PhoneNumber = number
            };

            try
            {
                var response = client.Publish(request);

                Console.WriteLine("Message sent to " + number + ":");
                Console.WriteLine(message);
            }
            catch (Exception ex)
            {
                Console.WriteLine("Caught exception publishing request:");
                Console.WriteLine(ex.Message);
            }
        }
    }
}
```

Lihat contoh lengkapnya, termasuk informasi tentang cara membuat dan menjalankan contoh dari baris perintah, pada GitHub.

Pesan menggunakan Amazon SQS

AWS SDK untuk .NET Mendukung Amazon Simple Queue Service (Amazon Simple Queue Service), yang merupakan layanan antrian pesan yang menangani pesan atau alur kerja antar komponen dalam sistem.

Antrian Amazon SQS menyediakan mekanisme yang memungkinkan Anda mengirim, menyimpan, dan menerima pesan antara komponen perangkat lunak seperti layanan mikro, sistem terdistribusi, dan aplikasi tanpa server. Ini memungkinkan Anda untuk memisahkan komponen tersebut dan

membebaskan Anda dari kebutuhan untuk merancang dan mengoperasikan sistem pesan Anda sendiri. Untuk informasi tentang cara kerja antrian dan pesan di Amazon SQS, lihat tutorial Amazon SQS dan arsitektur Amazon SQSDasar di Panduan Pengembang Layanan Antrian Sederhana Amazon.

> **Important**
>
> Karena sifat antrian yang terdistribusi, Amazon SQS tidak dapat menjamin bahwa Anda akan menerima pesan dalam urutan yang tepat yang dikirim. Jika Anda perlu mempertahankan urutan pesan, gunakan antrean Amazon SQS FIFO.

APIs

AWS SDK untuk .NET Menyediakan APIs untuk klien Amazon SQS. Ini APIs memungkinkan Anda untuk bekerja dengan fitur Amazon SQS seperti antrian dan pesan. Bagian ini berisi sejumlah kecil contoh yang menunjukkan pola yang dapat Anda ikuti saat bekerja dengan ini APIs. Untuk melihat set lengkap APIs, lihat Referensi AWS SDK untuk .NET API (dan gulir ke “Amazon.sqs”).

Amazon SQS APIs disediakan oleh AWSSDK NuGet paket.SQS.

Prasyarat

Sebelum Anda mulai, pastikan Anda telah mengatur lingkungan Anda dan mengkonfigurasi proyek Anda. Juga tinjau informasi diMenggunakan SDK .

Topik

Topik

- Membuat antrian Amazon SQS
- Memperbarui antrian Amazon SQS
- Menghapus antrian Amazon SQS
- Mengirim pesan Amazon SQS
- Menerima pesan Amazon SQS

Membuat antrian Amazon SQS

Contoh ini menunjukkan cara menggunakan antrean Amazon SQS AWS SDK untuk .NET untuk membuat antrean Amazon SQS. Aplikasi membuat antrian huruf mati jika Anda tidak menyediakan ARN untuk satu. Kemudian membuat antrian pesan standar, yang mencakup antrian huruf mati (yang Anda berikan atau yang dibuat).

Jika Anda tidak memberikan argumen baris perintah apa pun, aplikasi hanya menampilkan informasi tentang semua antrian yang ada.

Bagian berikut menyediakan cuplikan dari contoh ini. Kode lengkap untuk contoh ditampilkan setelah itu, dan dapat dibangun dan dijalankan apa adanya.

Topik

- Tampilkan antrian yang ada
- Buat antrian
- Dapatkan ARN antrian
- Kode lengkap
- Pertimbangan tambahan

Tampilkan antrian yang ada

Cuplikan berikut menunjukkan daftar antrian yang ada di wilayah klien SQS dan atribut setiap antrian.

Contoh di akhir topik ini menunjukkan cuplikan ini digunakan.

```
    //
    // Method to show a list of the existing queues
    private static async Task ShowQueues(IAmazonSQS sqsClient)
    {
      ListQueuesResponse responseList = await sqsClient.ListQueuesAsync("");
      Console.WriteLine();
      foreach(string qUrl in responseList.QueueUrls)
      {
        // Get and show all attributes. Could also get a subset.
        await ShowAllAttributes(sqsClient, qUrl);
      }
    }
```

```
    //
    // Method to show all attributes of a queue
    private static async Task ShowAllAttributes(IAmazonSQS sqsClient, string qUrl)
    {
      var attributes = new List<string>{ QueueAttributeName.All };
      GetQueueAttributesResponse responseGetAtt =
        await sqsClient.GetQueueAttributesAsync(qUrl, attributes);
      Console.WriteLine($"Queue: {qUrl}");
      foreach(var att in responseGetAtt.Attributes)
        Console.WriteLine($"\t{att.Key}: {att.Value}");
    }
```

Buat antrian

Cuplikan berikut membuat antrian. Cuplikan mencakup penggunaan antrian huruf mati, tetapi antrian huruf mati tidak selalu diperlukan untuk antrian Anda.

Contoh di akhir topik ini menunjukkan cuplikan ini digunakan.

```
    //
    // Method to create a queue. Returns the queue URL.
    private static async Task<string> CreateQueue(
      IAmazonSQS sqsClient, string qName, string deadLetterQueueUrl=null,
      string maxReceiveCount=null, string receiveWaitTime=null)
    {
      var attrs = new Dictionary<string, string>();

      // If a dead-letter queue is given, create a message queue
      if(!string.IsNullOrEmpty(deadLetterQueueUrl))
      {
        attrs.Add(QueueAttributeName.ReceiveMessageWaitTimeSeconds, receiveWaitTime);
        attrs.Add(QueueAttributeName.RedrivePolicy,
          $"{{\"deadLetterTargetArn\":\"{await GetQueueArn(sqsClient,
deadLetterQueueUrl)}\"," +
          $"\"maxReceiveCount\":\"{maxReceiveCount}\"}}");
        // Add other attributes for the message queue such as VisibilityTimeout
      }

      // If no dead-letter queue is given, create one of those instead
      //else
      //{
      //  // Add attributes for the dead-letter queue as needed
      //  attrs.Add();
```

```
      //}

      // Create the queue
      CreateQueueResponse responseCreate = await sqsClient.CreateQueueAsync(
          new CreateQueueRequest{QueueName = qName, Attributes = attrs});
      return responseCreate.QueueUrl;
    }
```

Dapatkan ARN antrian

Cuplikan berikut mendapatkan ARN dari antrian yang diidentifikasi oleh URL antrian yang diberikan.

Contoh di akhir topik ini menunjukkan cuplikan ini digunakan.

```
    //
    // Method to get the ARN of a queue
    private static async Task<string> GetQueueArn(IAmazonSQS sqsClient, string qUrl)
    {
      GetQueueAttributesResponse responseGetAtt = await
sqsClient.GetQueueAttributesAsync(
        qUrl, new List<string>{QueueAttributeName.QueueArn});
      return responseGetAtt.QueueARN;
    }
```

Kode lengkap

Bagian ini menunjukkan referensi yang relevan dan kode lengkap untuk contoh ini.

Referensi SDK

NuGet paket:

- AWSSDK.SQS

Elemen pemrograman:

- Namespace Amazon.sqs

  Kelas Amazon SQSClient

  Kelas QueueAttributeName

- Namespace Amazon.sqs.Model

  Kelas CreateQueueRequest

  Kelas CreateQueueResponse

  Kelas GetQueueAttributesResponse

  Kelas ListQueuesResponse

Kodenya

```
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSCreateQueue
{
  // = = =
```

```
      if(parsedArgs.Count == 0)
      {
        PrintHelp();
        Console.WriteLine("\nNo arguments specified.");
        Console.Write("Do you want to see a list of the existing queues? ((y) or n):
 ");
        string response = Console.ReadLine();
        if((string.IsNullOrEmpty(response)) || (response.ToLower() == "y"))
          await ShowQueues(sqsClient);
        return;
      }

      // Get the application arguments from the parsed list
      string queueName =
        CommandLine.GetArgument(parsedArgs, null, "-q", "--queue-name");
      string deadLetterQueueUrl =
        CommandLine.GetArgument(parsedArgs, null, "-d", "--dead-letter-queue");
      string maxReceiveCount =
        CommandLine.GetArgument(parsedArgs, MaxReceiveCount, "-m", "--max-receive-
count");
      string receiveWaitTime =
        CommandLine.GetArgument(parsedArgs, ReceiveMessageWaitTime, "-w", "--wait-
time");

      if(string.IsNullOrEmpty(queueName))
        CommandLine.ErrorExit(
          "\nYou must supply a queue name.\nRun the command with no arguments to see
 help.");

      // If a dead-letter queue wasn't given, create one
      if(string.IsNullOrEmpty(deadLetterQueueUrl))
      {
        Console.WriteLine("\nNo dead-letter queue was specified. Creating one...");
        deadLetterQueueUrl = await CreateQueue(sqsClient, queueName + "__dlq");
        Console.WriteLine($"Your new dead-letter queue:");
        await ShowAllAttributes(sqsClient, deadLetterQueueUrl);
      }

      // Create the message queue
      string messageQueueUrl = await CreateQueue(
        sqsClient, queueName, deadLetterQueueUrl, maxReceiveCount, receiveWaitTime);
      Console.WriteLine($"Your new message queue:");
      await ShowAllAttributes(sqsClient, messageQueueUrl);
    }
```

```
    //
    // Method to show a list of the existing queues
    private static async Task ShowQueues(IAmazonSQS sqsClient)
    {
      ListQueuesResponse responseList = await sqsClient.ListQueuesAsync("");
      Console.WriteLine();
      foreach(string qUrl in responseList.QueueUrls)
      {
        // Get and show all attributes. Could also get a subset.
        await ShowAllAttributes(sqsClient, qUrl);
      }
    }


    //
    // Method to create a queue. Returns the queue URL.
    private static async Task<string> CreateQueue(
      IAmazonSQS sqsClient, string qName, string deadLetterQueueUrl=null,
      string maxReceiveCount=null, string receiveWaitTime=null)
    {
      var attrs = new Dictionary<string, string>();

      // If a dead-letter queue is given, create a message queue
      if(!string.IsNullOrEmpty(deadLetterQueueUrl))
      {
        attrs.Add(QueueAttributeName.ReceiveMessageWaitTimeSeconds, receiveWaitTime);
        attrs.Add(QueueAttributeName.RedrivePolicy,
          $"{{\"deadLetterTargetArn\":\"{await GetQueueArn(sqsClient,
deadLetterQueueUrl)}\"," +
          $"\"maxReceiveCount\":\"{maxReceiveCount}\"}}");
        // Add other attributes for the message queue such as VisibilityTimeout
      }

      // If no dead-letter queue is given, create one of those instead
      //else
      //{
      //  // Add attributes for the dead-letter queue as needed
      //  attrs.Add();
      //}

      // Create the queue
      CreateQueueResponse responseCreate = await sqsClient.CreateQueueAsync(
```

```
          new CreateQueueRequest{QueueName = qName, Attributes = attrs});
      return responseCreate.QueueUrl;
    }


    //
    // Method to get the ARN of a queue
    private static async Task<string> GetQueueArn(IAmazonSQS sqsClient, string qUrl)
    {
      GetQueueAttributesResponse responseGetAtt = await
sqsClient.GetQueueAttributesAsync(
        qUrl, new List<string>{QueueAttributeName.QueueArn});
      return responseGetAtt.QueueARN;
    }


    //
    // Method to show all attributes of a queue
    private static async Task ShowAllAttributes(IAmazonSQS sqsClient, string qUrl)
    {
      var attributes = new List<string>{ QueueAttributeName.All };
      GetQueueAttributesResponse responseGetAtt =
        await sqsClient.GetQueueAttributesAsync(qUrl, attributes);
      Console.WriteLine($"Queue: {qUrl}");
      foreach(var att in responseGetAtt.Attributes)
        Console.WriteLine($"\t{att.Key}: {att.Value}");
    }


    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
      "\nUsage: SQSCreateQueue -q <queue-name> [-d <dead-letter-queue>]" +
        " [-m <max-receive-count>] [-w <wait-time>]" +
      "\n  -q, --queue-name: The name of the queue you want to create." +
      "\n  -d, --dead-letter-queue: The URL of an existing queue to be used as the
dead-letter queue."+
      "\n      If this argument isn't supplied, a new dead-letter queue will be
created." +
      "\n  -m, --max-receive-count: The value for maxReceiveCount in the RedrivePolicy
of the queue." +
      $"\n      Default is {MaxReceiveCount}." +
```

```
      "\n  -w, --wait-time: The value for ReceiveMessageWaitTimeSeconds of the queue
for long polling." +
      $"\n      Default is {ReceiveMessageWaitTime}.");
    }
  }


  // = = =
```

```
        // If the first argument in this iteration doesn't start with a dash, it's a
 value
        else
        {
          parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
          n++;
        }
      }

      return parsedArgs;
    }

    //
    // Method to get an argument from the parsed command-line arguments
    //
    // Parameters:
    // - parsedArgs: The Dictionary object returned from the Parse() method (shown
 above).
    // - defaultValue: The default string to return if the specified key isn't in
 parsedArgs.
    // - keys: An array of keys to look for in parsedArgs.
    public static string GetArgument(
      Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
    {
      string retval = null;
      foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
      return retval ?? defaultReturn;
    }

    //
    // Method to exit the application with an error.
    public static void ErrorExit(string msg, int code=1)
    {
      Console.WriteLine("\nError");
      Console.WriteLine(msg);
      Environment.Exit(code);
    }
  }

}
```

Pertimbangan tambahan

- Nama antrian Anda harus terdiri dari karakter alfanumerik, tanda hubung, dan garis bawah.
- Nama antrian dan antrian peka huruf URLs besar/kecil
- Jika Anda memerlukan URL antrian tetapi hanya memiliki nama antrian, gunakan salah satu metode. `AmazonSQSClient.GetQueueUrlAsync`
- Untuk informasi tentang berbagai atribut antrian yang dapat Anda atur, lihat CreateQueueRequestdi Referensi AWS SDK untuk .NET API atau SetQueueAttributesReferensi API Layanan Antrian Sederhana Amazon.
- Contoh ini menentukan polling panjang untuk semua pesan pada antrian yang Anda buat. Hal ini dilakukan dengan menggunakan `ReceiveMessageWaitTimeSeconds` atribut.

  Anda juga dapat menentukan polling panjang selama panggilan ke `ReceiveMessageAsync` metode SQSClient kelas Amazon. Untuk informasi selengkapnya, lihat Menerima pesan Amazon SQS.

  Untuk informasi tentang polling singkat versus polling panjang, lihat Pemungutan suara pendek dan panjang di Panduan Pengembang Layanan Antrian Sederhana Amazon.
- Antrian surat mati adalah antrian yang dapat ditargetkan oleh antrian (sumber) lain untuk pesan yang tidak berhasil diproses. Untuk informasi selengkapnya, lihat antrian surat mati Amazon SQS di Panduan Pengembang Layanan Antrian Sederhana Amazon.
- Anda juga dapat melihat daftar antrian dan hasil contoh ini di konsol Amazon SQS.

Memperbarui antrian Amazon SQS

Contoh ini menunjukkan cara menggunakan AWS SDK untuk .NET untuk memperbarui antrian Amazon SQS. Setelah beberapa pemeriksaan, aplikasi memperbarui atribut yang diberikan dengan nilai yang diberikan, dan kemudian menampilkan semua atribut untuk antrian.

Jika hanya URL antrian yang disertakan dalam argumen baris perintah, aplikasi hanya menampilkan semua atribut untuk antrian.

Bagian berikut menyediakan cuplikan dari contoh ini. Kode lengkap untuk contoh ditampilkan setelah itu, dan dapat dibangun dan dijalankan apa adanya.

Topik

- Tampilkan atribut antrian
- Validasi nama atribut
- Perbarui atribut antrian
- Kode lengkap
- Pertimbangan tambahan

Tampilkan atribut antrian

Cuplikan berikut menunjukkan atribut antrian yang diidentifikasi oleh URL antrian yang diberikan.

Contoh di akhir topik ini menunjukkan cuplikan ini digunakan.

```
//
// Method to show all attributes of a queue
private static async Task ShowAllAttributes(IAmazonSQS sqsClient, string qUrl)
{
  GetQueueAttributesResponse responseGetAtt =
    await sqsClient.GetQueueAttributesAsync(qUrl,
      new List<string>{ QueueAttributeName.All });
  Console.WriteLine($"Queue: {qUrl}");
  foreach(var att in responseGetAtt.Attributes)
    Console.WriteLine($"\t{att.Key}: {att.Value}");
}
```

Validasi nama atribut

Cuplikan berikut memvalidasi nama atribut yang sedang diperbarui.

Contoh di akhir topik ini menunjukkan cuplikan ini digunakan.

```
//
// Method to check the name of the attribute
private static bool ValidAttribute(string attribute)
```

```
    {
      var attOk = false;
      var qAttNameType = typeof(QueueAttributeName);
      List<string> qAttNamefields = new List<string>();
      foreach(var field in qAttNameType.GetFields())
       qAttNamefields.Add(field.Name);
      foreach(var name in qAttNamefields)
        if(attribute == name) { attOk = true; break; }
      return attOk;
    }
```

Perbarui atribut antrian

Cuplikan berikut memperbarui atribut antrian yang diidentifikasi oleh URL antrian yang diberikan.

Contoh di akhir topik ini menunjukkan cuplikan ini digunakan.

```
    //
    // Method to update a queue attribute
    private static async Task UpdateAttribute(
      IAmazonSQS sqsClient, string qUrl, string attribute, string value)
    {
      await sqsClient.SetQueueAttributesAsync(qUrl,
        new Dictionary<string, string>{{attribute, value}});
    }
```

Kode lengkap

Bagian ini menunjukkan referensi yang relevan dan kode lengkap untuk contoh ini.

Referensi SDK

NuGet paket:

- AWSSDK.SQS

Elemen pemrograman:

- Namespace Amazon.sqs

  Kelas Amazon SQSClient

  Kelas QueueAttributeName

- Namespace Amazon.sqs.Model

  Kelas GetQueueAttributesResponse

Kodenya

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSUpdateQueue
{
  // = = =
```

```
      // Create the Amazon SQS client
      var sqsClient = new AmazonSQSClient();

      // In the case of one command-line argument, just show the attributes for the
queue
      if(parsedArgs.Count == 1)
        await ShowAllAttributes(sqsClient, qUrl);

      // Otherwise, attempt to update the given queue attribute with the given value
      else
      {
        // Check to see if the attribute is valid
        if(ValidAttribute(attribute))
        {
          // Perform the update and then show all the attributes of the queue
          await UpdateAttribute(sqsClient, qUrl, attribute, value);
          await ShowAllAttributes(sqsClient, qUrl);
        }
        else
        {
          Console.WriteLine($"\nThe given attribute name, {attribute}, isn't valid.");
        }
      }
    }


    //
    // Method to show all attributes of a queue
    private static async Task ShowAllAttributes(IAmazonSQS sqsClient, string qUrl)
    {
      GetQueueAttributesResponse responseGetAtt =
        await sqsClient.GetQueueAttributesAsync(qUrl,
          new List<string>{ QueueAttributeName.All });
      Console.WriteLine($"Queue: {qUrl}");
      foreach(var att in responseGetAtt.Attributes)
        Console.WriteLine($"\t{att.Key}: {att.Value}");
    }


    //
    // Method to check the name of the attribute
    private static bool ValidAttribute(string attribute)
    {
```

```
      var attOk = false;
      var qAttNameType = typeof(QueueAttributeName);
      List<string> qAttNamefields = new List<string>();
      foreach(var field in qAttNameType.GetFields())
       qAttNamefields.Add(field.Name);
      foreach(var name in qAttNamefields)
        if(attribute == name) { attOk = true; break; }
      return attOk;
    }


    //
    // Method to update a queue attribute
    private static async Task UpdateAttribute(
      IAmazonSQS sqsClient, string qUrl, string attribute, string value)
    {
      await sqsClient.SetQueueAttributesAsync(qUrl,
        new Dictionary<string, string>{{attribute, value}});
    }


    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine("\nUsage: SQSUpdateQueue -q queue_url [-a attribute -v
value]");
      Console.WriteLine("  -q: The URL of the queue you want to update.");
      Console.WriteLine("  -a: The name of the attribute to update.");
      Console.WriteLine("  -v, --value: The value to assign to the attribute.");
    }
  }


  // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = =
  // Class that represents a command line on the console or terminal.
  // (This is the same for all examples. When you have seen it once, you can ignore
it.)
  static class CommandLine
  {
    //
    // Method to parse a command line of the form: "--key value" or "-k value".
    //
```

```
    // Parameters:
    // - args: The command-line arguments passed into the application by the system.
    //
    // Returns:
    // A Dictionary with string Keys and Values.
    //
    // If a key is found without a matching value, Dictionary.Value is set to the key
    //  (including the dashes).
    // If a value is found without a matching key, Dictionary.Key is set to "--NoKeyN",
    //  where "N" represents sequential numbers.
    public static Dictionary<string,string> Parse(string[] args)
    {
      var parsedArgs = new Dictionary<string,string>();
      int i = 0, n = 0;
      while(i < args.Length)
      {
        // If the first argument in this iteration starts with a dash it's an option.
        if(args[i].StartsWith("-"))
        {
          var key = args[i++];
          var value = key;

          // Check to see if there's a value that goes with this option?
          if((i < args.Length) && (!args[i].StartsWith("-"))) value = args[i++];
          parsedArgs.Add(key, value);
        }

        // If the first argument in this iteration doesn't start with a dash, it's a
value
        else
        {
          parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
          n++;
        }
      }

      return parsedArgs;
    }

    //
    // Method to get an argument from the parsed command-line arguments
    //
    // Parameters:
```

```
    // - parsedArgs: The Dictionary object returned from the Parse() method (shown
 above).
    // - defaultValue: The default string to return if the specified key isn't in
 parsedArgs.
    // - keys: An array of keys to look for in parsedArgs.
    public static string GetArgument(
      Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
    {
      string retval = null;
      foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
      return retval ?? defaultReturn;
    }

    //
    // Method to exit the application with an error.
    public static void ErrorExit(string msg, int code=1)
    {
      Console.WriteLine("\nError");
      Console.WriteLine(msg);
      Environment.Exit(code);
    }
  }

}
```

Pertimbangan tambahan

- Untuk memperbarui `RedrivePolicy` atribut, Anda harus mengutip seluruh nilai dan menghindari tanda kutip untuk key/value pasangan, yang sesuai untuk sistem operasi Anda.

  Pada Windows, misalnya, nilainya dibangun dengan cara yang mirip dengan yang berikut ini:

  ```
  "{\"deadLetterTargetArn\":\"DEAD_LETTER-QUEUE-ARN\",\"maxReceiveCount\":\"10\"}"
  ```

Menghapus antrian Amazon SQS

Contoh ini menunjukkan cara menggunakan antrean Amazon SQS AWS SDK untuk .NET untuk menghapus. Aplikasi menghapus antrian, menunggu hingga jumlah waktu tertentu untuk antrian hilang, dan kemudian menunjukkan daftar antrian yang tersisa.

Jika Anda tidak memberikan argumen baris perintah apa pun, aplikasi hanya menampilkan daftar antrian yang ada.

Bagian berikut menyediakan cuplikan dari contoh ini. Kode lengkap untuk contoh ditampilkan setelah itu, dan dapat dibangun dan dijalankan apa adanya.

Topik

- Hapus antrian
- Tunggu antrian hilang
- Tampilkan daftar antrian yang ada
- Kode lengkap
- Pertimbangan tambahan

Hapus antrian

Cuplikan berikut menghapus antrian yang diidentifikasi oleh URL antrian yang diberikan.

Contoh di akhir topik ini menunjukkan cuplikan ini digunakan.

```
//
// Method to delete an SQS queue
private static async Task DeleteQueue(IAmazonSQS sqsClient, string qUrl)
{
  Console.WriteLine($"Deleting queue {qUrl}...");
  await sqsClient.DeleteQueueAsync(qUrl);
  Console.WriteLine($"Queue {qUrl} has been deleted.");
}
```

Tunggu antrian hilang

Cuplikan berikut menunggu proses penghapusan selesai, yang mungkin memakan waktu 60 detik.

Contoh di akhir topik ini menunjukkan cuplikan ini digunakan.

```
//
// Method to wait up to a given number of seconds
private static async Task Wait(
  IAmazonSQS sqsClient, int numSeconds, string qUrl)
{
```

```
      Console.WriteLine($"Waiting for up to {numSeconds} seconds.");
      Console.WriteLine("Press any key to stop waiting. (Response might be slightly
delayed.)");
      for(int i=0; i<numSeconds; i++)
      {
        Console.Write(".");
        Thread.Sleep(1000);
        if(Console.KeyAvailable) break;

        // Check to see if the queue is gone yet
        var found = false;
        ListQueuesResponse responseList = await sqsClient.ListQueuesAsync("");
        foreach(var url in responseList.QueueUrls)
        {
          if(url == qUrl)
          {
            found = true;
            break;
          }
        }
        if(!found) break;
      }
    }
```

Tampilkan daftar antrian yang ada

Cuplikan berikut menunjukkan daftar antrian yang ada di wilayah klien SQS.

Contoh di akhir topik ini menunjukkan cuplikan ini digunakan.

```
    //
    // Method to show a list of the existing queues
    private static async Task ListQueues(IAmazonSQS sqsClient)
    {
      ListQueuesResponse responseList = await sqsClient.ListQueuesAsync("");
      Console.WriteLine("\nList of queues:");
      foreach(var qUrl in responseList.QueueUrls)
        Console.WriteLine($"- {qUrl}");
    }
```

Kode lengkap

Bagian ini menunjukkan referensi yang relevan dan kode lengkap untuk contoh ini.

Referensi SDK

NuGet paket:

- AWSSDK.SQS

Elemen pemrograman:

- Namespace Amazon.sqs

  Kelas Amazon SQSClient
- Namespace Amazon.sqs.Model

  Kelas ListQueuesResponse

Kodenya

```
using System;
using System.Threading;
using System.Threading.Tasks;
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSDeleteQueue
{
  // = = =
```

```
        Console.WriteLine("\nNo arguments specified.");
        Console.Write("Do you want to see a list of the existing queues? ((y) or n):
");
        var response = Console.ReadLine();
        if((string.IsNullOrEmpty(response)) || (response.ToLower() == "y"))
          await ListQueues(sqsClient);
        return;
      }

      // If given a queue URL, delete that queue
      if(args[0].StartsWith("https://sqs."))
      {
        // Delete the queue
        await DeleteQueue(sqsClient, args[0]);
        // Wait for a little while because it takes a while for the queue to disappear
        await Wait(sqsClient, TimeToWait, args[0]);
        // Show a list of the remaining queues
        await ListQueues(sqsClient);
      }
      else
      {
        Console.WriteLine("The command-line argument isn't a queue URL:");
        Console.WriteLine($"{args[0]}");
      }
    }


    //
    // Method to delete an SQS queue
    private static async Task DeleteQueue(IAmazonSQS sqsClient, string qUrl)
    {
      Console.WriteLine($"Deleting queue {qUrl}...");
      await sqsClient.DeleteQueueAsync(qUrl);
      Console.WriteLine($"Queue {qUrl} has been deleted.");
    }


    //
    // Method to wait up to a given number of seconds
    private static async Task Wait(
      IAmazonSQS sqsClient, int numSeconds, string qUrl)
    {
      Console.WriteLine($"Waiting for up to {numSeconds} seconds.");
```

```
      Console.WriteLine("Press any key to stop waiting. (Response might be slightly
 delayed.)");
      for(int i=0; i<numSeconds; i++)
      {
        Console.Write(".");
        Thread.Sleep(1000);
        if(Console.KeyAvailable) break;

        // Check to see if the queue is gone yet
        var found = false;
        ListQueuesResponse responseList = await sqsClient.ListQueuesAsync("");
        foreach(var url in responseList.QueueUrls)
        {
          if(url == qUrl)
          {
            found = true;
            break;
          }
        }
        if(!found) break;
      }
    }


    //
    // Method to show a list of the existing queues
    private static async Task ListQueues(IAmazonSQS sqsClient)
    {
      ListQueuesResponse responseList = await sqsClient.ListQueuesAsync("");
      Console.WriteLine("\nList of queues:");
      foreach(var qUrl in responseList.QueueUrls)
        Console.WriteLine($"- {qUrl}");
    }
  }
}
```

Pertimbangan tambahan

- Panggilan `DeleteQueueAsync` API tidak memeriksa untuk melihat apakah antrian yang Anda hapus digunakan sebagai antrian huruf mati. Prosedur yang lebih canggih dapat memeriksa ini.

- Anda juga dapat melihat daftar antrian dan hasil contoh ini di konsol Amazon SQS.

Mengirim pesan Amazon SQS

Contoh ini menunjukkan kepada Anda cara menggunakan untuk mengirim pesan AWS SDK untuk .NET ke antrean Amazon SQS, yang dapat Anda buat secara terprogram atau dengan menggunakan konsol Amazon SQS. Aplikasi mengirimkan satu pesan ke antrian dan kemudian sekumpulan pesan. Aplikasi kemudian menunggu input pengguna, yang dapat berupa pesan tambahan untuk dikirim ke antrian atau permintaan untuk keluar dari aplikasi.

Contoh ini dan contoh selanjutnya tentang menerima pesan dapat digunakan bersama untuk melihat aliran pesan di Amazon SQS.

Bagian berikut menyediakan cuplikan dari contoh ini. Kode lengkap untuk contoh ditampilkan setelah itu, dan dapat dibangun dan dijalankan apa adanya.

Topik

- Kirim pesan
- Kirim sejumlah pesan
- Hapus semua pesan dari antrian
- Kode lengkap
- Pertimbangan tambahan

Kirim pesan

Cuplikan berikut mengirimkan pesan ke antrian yang diidentifikasi oleh URL antrian yang diberikan.

Contoh di akhir topik ini menunjukkan cuplikan ini digunakan.

```
//
// Method to put a message on a queue
// Could be expanded to include message attributes, etc., in a SendMessageRequest
private static async Task SendMessage(
  IAmazonSQS sqsClient, string qUrl, string messageBody)
{
  SendMessageResponse responseSendMsg =
    await sqsClient.SendMessageAsync(qUrl, messageBody);
  Console.WriteLine($"Message added to queue\n  {qUrl}");
  Console.WriteLine($"HttpStatusCode: {responseSendMsg.HttpStatusCode}");
}
```

Kirim sejumlah pesan

Cuplikan berikut mengirimkan sekumpulan pesan ke antrian yang diidentifikasi oleh URL antrian yang diberikan.

Contoh di akhir topik ini menunjukkan cuplikan ini digunakan.

```
//
// Method to put a batch of messages on a queue
// Could be expanded to include message attributes, etc.,
// in the SendMessageBatchRequestEntry objects
private static async Task SendMessageBatch(
  IAmazonSQS sqsClient, string qUrl, List<SendMessageBatchRequestEntry> messages)
{
  Console.WriteLine($"\nSending a batch of messages to queue\n  {qUrl}");
  SendMessageBatchResponse responseSendBatch =
    await sqsClient.SendMessageBatchAsync(qUrl, messages);
  // Could test responseSendBatch.Failed here
  foreach(SendMessageBatchResultEntry entry in responseSendBatch.Successful)
    Console.WriteLine($"Message {entry.Id} successfully queued.");
}
```

Hapus semua pesan dari antrian

Cuplikan berikut menghapus semua pesan dari antrian yang diidentifikasi oleh URL antrian yang diberikan. Ini juga dikenal sebagai membersihkan antrian.

Contoh di akhir topik ini menunjukkan cuplikan ini digunakan.

```
//
// Method to delete all messages from the queue
private static async Task DeleteAllMessages(IAmazonSQS sqsClient, string qUrl)
{
  Console.WriteLine($"\nPurging messages from queue\n  {qUrl}...");
  PurgeQueueResponse responsePurge = await sqsClient.PurgeQueueAsync(qUrl);
  Console.WriteLine($"HttpStatusCode: {responsePurge.HttpStatusCode}");
}
```

Kode lengkap

Bagian ini menunjukkan referensi yang relevan dan kode lengkap untuk contoh ini.

Referensi SDK

NuGet paket:

- AWSSDK.SQS

Elemen pemrograman:

- Namespace Amazon.sqs

  Kelas Amazon SQSClient
- Namespace Amazon.sqs.Model

  Kelas PurgeQueueResponse

  Kelas SendMessageBatchResponse

  Kelas SendMessageResponse

  Kelas SendMessageBatchRequestEntry

  Kelas SendMessageBatchResultEntry

Kodenya

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSSendMessages
{
  // = = =
```

```
    private const string XmlMessage = "<products><product name=\"Product A\" price=
\"32\" /><product name=\"Product B\" price=\"27\" /></products>";
    private const string CustomMessage = "||product|Product A|32||product|Product B|
27||";
    private const string TextMessage = "Just a plain text message.";

    static async Task Main(string[] args)
    {
      // Do some checks on the command-line
      if(args.Length == 0)
      {
        Console.WriteLine("\nUsage: SQSSendMessages queue_url");
        Console.WriteLine("   queue_url - The URL of an existing SQS queue.");
        return;
      }
      if(!args[0].StartsWith("https://sqs."))
      {
        Console.WriteLine("\nThe command-line argument isn't a queue URL:");
        Console.WriteLine($"{args[0]}");
        return;
      }

      // Create the Amazon SQS client
      var sqsClient = new AmazonSQSClient();

      // (could verify that the queue exists)
      // Send some example messages to the given queue
      // A single message
      await SendMessage(sqsClient, args[0], JsonMessage);

      // A batch of messages
      var batchMessages = new List<SendMessageBatchRequestEntry>{
        new SendMessageBatchRequestEntry("xmlMsg", XmlMessage),
        new SendMessageBatchRequestEntry("customeMsg", CustomMessage),
        new SendMessageBatchRequestEntry("textMsg", TextMessage)};
      await SendMessageBatch(sqsClient, args[0], batchMessages);

      // Let the user send their own messages or quit
      await InteractWithUser(sqsClient, args[0]);

      // Delete all messages that are still in the queue
      await DeleteAllMessages(sqsClient, args[0]);
    }
```

```
    //
    // Method to put a message on a queue
    // Could be expanded to include message attributes, etc., in a SendMessageRequest
    private static async Task SendMessage(
      IAmazonSQS sqsClient, string qUrl, string messageBody)
    {
      SendMessageResponse responseSendMsg =
        await sqsClient.SendMessageAsync(qUrl, messageBody);
      Console.WriteLine($"Message added to queue\n  {qUrl}");
      Console.WriteLine($"HttpStatusCode: {responseSendMsg.HttpStatusCode}");
    }


    //
    // Method to put a batch of messages on a queue
    // Could be expanded to include message attributes, etc.,
    // in the SendMessageBatchRequestEntry objects
    private static async Task SendMessageBatch(
      IAmazonSQS sqsClient, string qUrl, List<SendMessageBatchRequestEntry> messages)
    {
      Console.WriteLine($"\nSending a batch of messages to queue\n  {qUrl}");
      SendMessageBatchResponse responseSendBatch =
        await sqsClient.SendMessageBatchAsync(qUrl, messages);
      // Could test responseSendBatch.Failed here
      foreach(SendMessageBatchResultEntry entry in responseSendBatch.Successful)
        Console.WriteLine($"Message {entry.Id} successfully queued.");
    }


    //
    // Method to get input from the user
    // They can provide messages to put in the queue or exit the application
    private static async Task InteractWithUser(IAmazonSQS sqsClient, string qUrl)
    {
      string response;
      while (true)
      {
        // Get the user's input
        Console.WriteLine("\nType a message for the queue or \"exit\" to quit:");
        response = Console.ReadLine();
        if(response.ToLower() == "exit") break;

        // Put the user's message in the queue
```

```
        await SendMessage(sqsClient, qUrl, response);
      }
    }


    //
    // Method to delete all messages from the queue
    private static async Task DeleteAllMessages(IAmazonSQS sqsClient, string qUrl)
    {
      Console.WriteLine($"\nPurging messages from queue\n  {qUrl}...");
      PurgeQueueResponse responsePurge = await sqsClient.PurgeQueueAsync(qUrl);
      Console.WriteLine($"HttpStatusCode: {responsePurge.HttpStatusCode}");
    }
  }
}
```

Pertimbangan tambahan

- Untuk informasi tentang berbagai batasan pada pesan, termasuk karakter yang diizinkan, lihat Kuota yang terkait dengan pesan di Panduan Pengembang Layanan Antrian Sederhana Amazon.

- Pesan tetap dalam antrian sampai dihapus atau antrian dibersihkan. Ketika pesan telah diterima oleh aplikasi, itu tidak akan terlihat dalam antrian meskipun masih ada dalam antrian. Untuk informasi selengkapnya tentang batas waktu visibilitas, lihat batas waktu visibilitas Amazon SQS.

- Selain isi pesan, Anda juga dapat menambahkan atribut ke pesan. Untuk informasi selengkapnya, lihat Metadata pesan.

Menerima pesan Amazon SQS

Contoh ini menunjukkan cara menggunakan AWS SDK untuk .NET untuk menerima pesan dari antrean Amazon SQS, yang dapat Anda buat secara terprogram atau menggunakan konsol Amazon SQS. Aplikasi membaca satu pesan dari antrian, memproses pesan (dalam hal ini, menampilkan badan pesan di konsol), dan kemudian menghapus pesan dari antrian. Aplikasi mengulangi langkah-langkah ini sampai pengguna mengetik tombol pada keyboard.

Contoh ini dan contoh sebelumnya tentang mengirim pesan dapat digunakan bersama untuk melihat aliran pesan di Amazon SQS.

Bagian berikut menyediakan cuplikan dari contoh ini. Kode lengkap untuk contoh ditampilkan setelah itu, dan dapat dibangun dan dijalankan apa adanya.

Topik

- Menerima pesan
- Menghapus pesan
- Kode lengkap
- Pertimbangan tambahan

Menerima pesan

Cuplikan berikut menerima pesan dari antrian yang diidentifikasi oleh URL antrian yang diberikan.

Contoh di akhir topik ini menunjukkan cuplikan ini digunakan.

```
//
// Method to read a message from the given queue
// In this example, it gets one message at a time
private static async Task<ReceiveMessageResponse> GetMessage(
  IAmazonSQS sqsClient, string qUrl, int waitTime=0)
{
  return await sqsClient.ReceiveMessageAsync(new ReceiveMessageRequest{
    QueueUrl=qUrl,
    MaxNumberOfMessages=MaxMessages,
    WaitTimeSeconds=waitTime
    // (Could also request attributes, set visibility timeout, etc.)
  });
}
```

Menghapus pesan

Cuplikan berikut menghapus pesan dari antrian yang diidentifikasi oleh URL antrian yang diberikan.

Contoh di akhir topik ini menunjukkan cuplikan ini digunakan.

```
//
// Method to delete a message from a queue
private static async Task DeleteMessage(
  IAmazonSQS sqsClient, Message message, string qUrl)
{
  Console.WriteLine($"\nDeleting message {message.MessageId} from queue...");
```

```
      await sqsClient.DeleteMessageAsync(qUrl, message.ReceiptHandle);
    }
```

Kode lengkap

Bagian ini menunjukkan referensi yang relevan dan kode lengkap untuk contoh ini.

Referensi SDK

NuGet paket:

- AWSSDK.SQS

Elemen pemrograman:

- Namespace Amazon.sqs

  Kelas Amazon SQSClient
- Namespace Amazon.sqs.Model

  Kelas ReceiveMessageRequest

  Kelas ReceiveMessageResponse

Kodenya

```
using System;
using System.Threading.Tasks;
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSReceiveMessages
{
  class Program
  {
    private const int MaxMessages = 1;
    private const int WaitTime = 2;
    static async Task Main(string[] args)
    {
      // Do some checks on the command-line
      if(args.Length == 0)
      {
```

```
        Console.WriteLine("\nUsage: SQSReceiveMessages queue_url");
        Console.WriteLine("   queue_url - The URL of an existing SQS queue.");
        return;
      }
      if(!args[0].StartsWith("https://sqs."))
      {
        Console.WriteLine("\nThe command-line argument isn't a queue URL:");
        Console.WriteLine($"{args[0]}");
        return;
      }

      // Create the Amazon SQS client
      var sqsClient = new AmazonSQSClient();

      // (could verify that the queue exists)
      // Read messages from the queue and perform appropriate actions
      Console.WriteLine($"Reading messages from queue\n  {args[0]}");
      Console.WriteLine("Press any key to stop. (Response might be slightly
delayed.)");
      do
      {
        var msg = await GetMessage(sqsClient, args[0], WaitTime);
        if(msg.Messages.Count != 0)
        {
          if(ProcessMessage(msg.Messages[0]))
            await DeleteMessage(sqsClient, msg.Messages[0], args[0]);
        }
      } while(!Console.KeyAvailable);
    }


    //
    // Method to read a message from the given queue
    // In this example, it gets one message at a time
    private static async Task<ReceiveMessageResponse> GetMessage(
      IAmazonSQS sqsClient, string qUrl, int waitTime=0)
    {
      return await sqsClient.ReceiveMessageAsync(new ReceiveMessageRequest{
        QueueUrl=qUrl,
        MaxNumberOfMessages=MaxMessages,
        WaitTimeSeconds=waitTime
        // (Could also request attributes, set visibility timeout, etc.)
      });
    }
```

```
    //
    // Method to process a message
    // In this example, it simply prints the message
    private static bool ProcessMessage(Message message)
    {
      Console.WriteLine($"\nMessage body of {message.MessageId}:");
      Console.WriteLine($"{message.Body}");
      return true;
    }


    //
    // Method to delete a message from a queue
    private static async Task DeleteMessage(
      IAmazonSQS sqsClient, Message message, string qUrl)
    {
      Console.WriteLine($"\nDeleting message {message.MessageId} from queue...");
      await sqsClient.DeleteMessageAsync(qUrl, message.ReceiptHandle);
    }
  }
}
```

Pertimbangan tambahan

- Untuk menentukan polling panjang, contoh ini menggunakan `WaitTimeSeconds` properti untuk setiap panggilan ke `ReceiveMessageAsync` metode.

  Anda juga dapat menentukan polling panjang untuk semua pesan pada antrian dengan menggunakan `ReceiveMessageWaitTimeSeconds` atribut saat membuat atau memperbarui antrian.

  Untuk informasi tentang polling singkat versus polling panjang, lihat Pemungutan suara pendek dan panjang di Panduan Pengembang Layanan Antrian Sederhana Amazon.

- Selama pemrosesan pesan, Anda dapat menggunakan tanda terima untuk mengubah batas waktu visibilitas pesan. Untuk informasi tentang cara melakukannya, lihat `ChangeMessageVisibilityAsync` metode SQSClient kelas Amazon.

- Memanggil `DeleteMessageAsync` metode tanpa syarat akan menghapus pesan dari antrian, terlepas dari pengaturan batas waktu visibilitas.

Menggunakan AWS Lambda untuk layanan komputasi

AWS SDK untuk .NET Dukungan AWS Lambda, yang memungkinkan Anda menjalankan kode tanpa menyediakan atau mengelola server. Untuk informasi selengkapnya, lihat halaman AWS Lambda produk dan Panduan AWS Lambda Pengembang, khususnya bagian untuk Bekerja dengan C#.

APIs

AWS SDK untuk .NET Menyediakan APIs untuk AWS Lambda. Ini APIs memungkinkan Anda untuk bekerja dengan fitur Lambda seperti fungsi, pemicu, dan peristiwa. Untuk melihat set lengkap APIs, lihat Lambda di Referensi AWS SDK untuk .NET API.

Lambda APIs disediakan oleh NuGet paket.

Prasyarat

Sebelum Anda mulai, pastikan Anda telah mengatur lingkungan Anda dan mengkonfigurasi proyek Anda. Juga tinjau informasi diMenggunakan SDK .

Informasi tambahan

Lihat Integrasi AWS dengan .NET Aspire untuk informasi tentang mengembangkan dengan AWS Lambda melalui .NET Aspire.

Topik

Topik

- Menggunakan anotasi untuk menulis fungsi AWS Lambda

Menggunakan anotasi untuk menulis fungsi AWS Lambda

Saat menulis fungsi Lambda, Anda terkadang perlu menulis sejumlah besar kode handler dan memperbarui AWS CloudFormation template, di antara tugas-tugas lainnya. Anotasi Lambda adalah kerangka kerja untuk membantu meringankan beban ini untuk fungsi.NET 6 Lambda, sehingga membuat pengalaman menulis Lambda terasa lebih alami di C #.

Sebagai contoh manfaat menggunakan kerangka kerja Anotasi Lambda, pertimbangkan cuplikan kode berikut yang menambahkan dua angka.

Tanpa Anotasi Lambda

```
public class Functions
{
    public APIGatewayProxyResponse LambdaMathPlus(APIGatewayProxyRequest request,
 ILambdaContext context)
    {
        if (!request.PathParameters.TryGetValue("x", out var xs))
        {
            return new APIGatewayProxyResponse
            {
                StatusCode = (int)HttpStatusCode.BadRequest
            };
        }
        if (!request.PathParameters.TryGetValue("y", out var ys))
        {
            return new APIGatewayProxyResponse
            {
                StatusCode = (int)HttpStatusCode.BadRequest
            };
        }

        var x = int.Parse(xs);
        var y = int.Parse(ys);

        return new APIGatewayProxyResponse
        {
            StatusCode = (int)HttpStatusCode.OK,
            Body = (x + y).ToString(),
            Headers = new Dictionary<string, string> { { "Content-Type", "text/
plain" } }
        };
    }
}
```

Dengan Anotasi Lambda

```
public class Functions
{
    [LambdaFunction]
```

```
    [RestApi("/plus/{x}/{y}")]
    public int Plus(int x, int y)
    {
        return x + y;
    }
}
```

Seperti yang ditunjukkan pada contoh, Anotasi Lambda dapat menghilangkan kebutuhan akan kode pelat boiler tertentu.

Untuk detail tentang cara menggunakan kerangka kerja serta informasi tambahan, lihat sumber daya berikut:

- GitHub README untuk dokumentasi tentang APIs dan atribut Anotasi Lambda.
- Posting blog untuk Anotasi Lambda.
- Paket `Amazon.Lambda.Annotations` NuGet .
- Proyek Photo Asset Management di GitHub. Secara khusus, lihat PamApiAnnotationsfolder dan referensi ke Anotasi Lambda di proyek README.

> **Note**
>
> Contoh sebelumnya khusus untuk V3 dari. AWS SDK untuk .NET Jika Anda menggunakan contoh dengan V4 SDK (versi terbaru), Anda mungkin perlu melakukan penyesuaian sesuai dengan informasi di. Migrasi ke versi 4

Pustaka dan kerangka kerja tingkat tinggi untuk AWS SDK untuk .NET

Bagian berikut berisi informasi tentang pustaka dan kerangka kerja tingkat tinggi yang bukan bagian dari fungsionalitas SDK inti. Pustaka dan kerangka kerja ini menggunakan fungsionalitas SDK inti untuk membuat fitur yang memudahkan tugas tertentu.

Jika Anda baru mengenal AWS SDK untuk .NET, Anda mungkin ingin memeriksa Membuat aplikasi sederhana topiknya terlebih dahulu. Ini memberi Anda pengantar SDK.

Sebelum Anda mulai, pastikan Anda telah mengatur lingkungan Anda dan mengkonfigurasi proyek Anda. Juga tinjau informasi diMenggunakan SDK .

Topik

- AWS Kerangka Pemrosesan Pesan untuk.NET
- Integrasi AWS dengan .NET Aspire di AWS SDK untuk .NET

AWS Kerangka Pemrosesan Pesan untuk.NET

AWS Message Processing Framework untuk.NET adalah framework AWS-native yang menyederhanakan pengembangan aplikasi pemrosesan pesan.NET yang menggunakan AWS layanan seperti Amazon Simple Queue Service (SQS), Amazon Simple Notification Service (SNS), dan Amazon. EventBridge Kerangka kerja ini mengurangi jumlah kode boiler-plate yang perlu ditulis oleh pengembang, memungkinkan Anda untuk fokus pada logika bisnis Anda saat menerbitkan dan mengonsumsi pesan. Untuk detail tentang bagaimana kerangka kerja dapat menyederhanakan pengembangan Anda, lihat posting blog Memperkenalkan Kerangka Pemrosesan AWS Pesan untuk.NET (Pratinjau). Bagian pertama khususnya menyediakan demonstrasi yang menunjukkan perbedaan antara menggunakan panggilan API tingkat rendah dan menggunakan kerangka kerja.

Kerangka Pemrosesan Pesan mendukung aktivitas dan fitur berikut:

- Mengirim pesan ke SQS dan menerbitkan acara ke SNS dan. EventBridge
- Menerima dan menangani pesan dari SQS dengan menggunakan poller yang berjalan lama, yang biasanya digunakan dalam layanan latar belakang. Ini termasuk mengelola batas waktu visibilitas saat pesan ditangani untuk mencegah klien lain memprosesnya.
- Menangani pesan dalam AWS Lambda fungsi.
- FIFO (first-in-first-out) SQS antrian dan topik SNS.
- OpenTelemetry untuk penebangan.

Untuk detail tentang aktivitas dan fitur ini, lihat bagian Fitur dari posting blog dan topik yang tercantum di bawah ini.

Sebelum Anda mulai, pastikan Anda telah mengatur lingkungan Anda dan mengkonfigurasi proyek Anda. Juga tinjau informasi diMenggunakan SDK .

Sumber daya tambahan

- `AWS.Messaging`Paket di NuGet.org.
- Referensi API.

- `README`File di GitHub repo di https://github.com/aws/aws-dotnet-messaging/
- .NET Injeksi ketergantungan.NET dari Microsoft.
- .NET Generic Host dari Microsoft.

Topik

- Memulai dengan AWS Message Processing Framework untuk.NET
- Publikasikan pesan dengan AWS Message Processing Framework untuk.NET
- Mengkonsumsi pesan dengan AWS Message Processing Framework untuk.NET
- Menggunakan FIFO dengan AWS Message Processing Framework untuk.NET
- Logging dan Buka Telemetri untuk Kerangka Pemrosesan AWS Pesan untuk.NET
- Kustomisasi Kerangka Pemrosesan AWS Pesan untuk.NET
- Keamanan untuk Kerangka Pemrosesan AWS Pesan untuk.NET

Memulai dengan AWS Message Processing Framework untuk.NET

Sebelum Anda mulai, pastikan Anda telah mengatur lingkungan Anda dan mengkonfigurasi proyek Anda. Juga tinjau informasi diMenggunakan SDK .

Topik ini memberikan informasi yang akan membantu Anda mulai menggunakan Kerangka Pemrosesan Pesan. Selain informasi prasyarat dan konfigurasi, tutorial disediakan yang menunjukkan kepada Anda bagaimana menerapkan skenario umum.

Prasyarat dan konfigurasi

- Kredensyal yang Anda berikan untuk aplikasi Anda harus memiliki izin yang sesuai untuk layanan pesan dan operasi yang digunakannya. Untuk informasi selengkapnya, lihat topik keamanan untuk SQS, SNS, dan EventBridgedi panduan pengembang masing-masing. Lihat juga bagian file README GitHub yang membahas izin tertentu.
- Untuk menggunakan AWS Message Processing Framework untuk.NET, Anda harus menambahkan `AWS.Messaging` NuGetpaket ke proyek Anda. Misalnya:

```
dotnet add package AWS.Messaging
```

- Kerangka kerja terintegrasi dengan wadah servis dependency injection (DI) .NET. Anda dapat mengonfigurasi kerangka kerja selama startup aplikasi Anda dengan menelepon `AddAWSMessageBus` untuk menambahkannya ke wadah DI.

```
var builder = WebApplication.CreateBuilder(args);

// Register the AWS Message Processing Framework for .NET
builder.Services.AddAWSMessageBus(builder =>
{
    // Register that you'll publish messages of type ChatMessage to an existing queue
    builder.AddSQSPublisher<ChatMessage>("https://sqs.us-
west-2.amazonaws.com/012345678910/MyAppProd");
});
```

Tutorial

Tutorial ini menunjukkan bagaimana menggunakan AWS Message Processing Framework untuk.NET. Ini menciptakan dua aplikasi: ASP.NET Core Minimal API yang mengirim pesan ke antrian Amazon SQS saat menerima permintaan di titik akhir API, dan aplikasi konsol yang berjalan lama yang melakukan polling untuk pesan-pesan ini dan menanganinya.

- Petunjuk dalam tutorial ini mendukung CLI .NET, tetapi Anda dapat melakukan tutorial ini dengan menggunakan alat lintas platform seperti .NET CLI atau Microsoft Visual Studio. Untuk informasi tentang alat, lihatMenginstal dan mengonfigurasi toolchain Anda untuk AWS SDK untuk .NET.
- Tutorial ini mengasumsikan bahwa Anda menggunakan `[default]` profil Anda untuk kredensyal. Ini juga mengasumsikan bahwa kredensyal jangka pendek tersedia dengan izin yang sesuai untuk mengirim dan menerima pesan Amazon SQS. Untuk informasi selengkapnya, lihat Mengautentikasi dengan AWS SDK untuk .NET AWS dan topik keamanan untuk SQS.

> **Note**
>
> Dengan menjalankan tutorial ini, Anda mungkin dikenakan biaya untuk pesan SQS.

Langkah-langkah

- Buat antrian SQS
- Membuat dan menjalankan aplikasi penerbitan
- Buat dan jalankan aplikasi penanganan
- Pembersihan

Buat antrian SQS

Tutorial ini membutuhkan antrian SQS untuk mengirim pesan ke dan menerima pesan dari. Antrian dapat dibuat dengan menggunakan salah satu perintah berikut untuk AWS CLI atau. Alat AWS untuk PowerShell Perhatikan URL antrian yang dikembalikan sehingga Anda dapat menentukannya dalam konfigurasi kerangka kerja berikut.

AWS CLI

```
aws sqs create-queue --queue-name DemoQueue
```

Alat AWS untuk PowerShell

```
New-SQSQueue -QueueName DemoQueue
```

Membuat dan menjalankan aplikasi penerbitan

Gunakan prosedur berikut untuk membuat dan menjalankan aplikasi penerbitan.

1. Buka command prompt atau terminal. Temukan atau buat folder sistem operasi di mana Anda dapat membuat proyek.NET.
2. Dalam folder itu, jalankan perintah berikut untuk membuat proyek.NET.

   ```
   dotnet new webapi --name Publisher
   ```
3. Arahkan ke folder proyek baru. Tambahkan ketergantungan pada AWS Message Processing Framework untuk.NET.

   ```
   cd Publisher
   dotnet add package AWS.Messaging
   ```

   > **Note**
   >
   > Jika Anda menggunakan AWS IAM Identity Center untuk otentikasi, pastikan untuk juga menambahkan `AWSSDK.SSO` dan`AWSSDK.SSOOIDC`.
4. Ganti kode `Program.cs` dengan kode berikut.

   ```
   using AWS.Messaging;
   ```

```
using Microsoft.AspNetCore.Mvc;
using Publisher;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/
swashbuckle.
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();


// Configure the AWS Message Processing Framework for .NET.
builder.Services.AddAWSMessageBus(builder =>
{
    // Check for input SQS URL.
    // The SQS URL should be passed as a command line argument or set in the Debug
 launch profile.
    if ((args.Length == 1) && (args[0].Contains("https://sqs.")))
    {
        // Register that you'll publish messages of type GreetingMessage:
        // 1. To a specified queue.
        // 2. Using the message identifier "greetingMessage", which will be used
        //    by handlers to route the message to the appropriate handler.
        builder.AddSQSPublisher<GreetingMessage>(args[0], "greetingMessage");
    }
    // You can map additional message types to queues or topics here as well.
});
var app = builder.Build();


// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

// Create an API Endpoint that receives GreetingMessage objects
// from the caller and then sends them as an SQS message.
app.MapPost("/greeting", async ([FromServices] IMessagePublisher publisher,
 Publisher.GreetingMessage message) =>
```

```
    {
        return await PostGreeting(message, publisher);
    })
.WithName("SendGreeting")
.WithOpenApi();

app.Run();

public partial class Program
{
    /// <summary>
    /// Endpoint for posting a greeting message.
    /// </summary>
    /// <param name="greetingMessage">The greeting message.</param>
    /// <param name="messagePublisher">The message publisher.</param>
    /// <returns>Async task result.</returns>
    public static async Task<IResult> PostGreeting(GreetingMessage greetingMessage,
        IMessagePublisher messagePublisher)
    {
        if (greetingMessage.SenderName == null || greetingMessage.Greeting == null)
        {
            return Results.BadRequest();
        }

        // Publish the message to the queue configured above.
        await messagePublisher.PublishAsync(greetingMessage);

        return Results.Ok();
    }
}

namespace Publisher
{
    /// <summary>
    /// This class represents the message contents.
    /// </summary>
    public class GreetingMessage
    {
        public string? SenderName { get; set; }
        public string? Greeting { get; set; }
    }
}
```

5. Jalankan perintah berikut. Ini akan membuka jendela browser dengan UI Swagger, yang memungkinkan Anda menjelajahi dan menguji API Anda.

```
dotnet watch run <queue URL created earlier>
```

6. Buka `/greeting` endpoint dan pilih Try it out.
7. Tentukan `senderName` dan `greeting` nilai untuk pesan, dan pilih Jalankan. Ini memanggil API Anda, yang mengirimkan pesan SQS.

Buat dan jalankan aplikasi penanganan

Gunakan prosedur berikut untuk membuat dan menjalankan aplikasi penanganan.

1. Buka command prompt atau terminal. Temukan atau buat folder sistem operasi di mana Anda dapat membuat proyek.NET.
2. Dalam folder itu, jalankan perintah berikut untuk membuat proyek.NET.

```
dotnet new console --name Handler
```

3. Arahkan ke folder proyek baru. Tambahkan ketergantungan pada AWS Message Processing Framework untuk.NET. Tambahkan juga `Microsoft.Extensions.Hosting` paket, yang memungkinkan Anda untuk mengkonfigurasi kerangka kerja melalui .NET Generic Host.

```
cd Handler
dotnet add package AWS.Messaging
dotnet add package Microsoft.Extensions.Hosting
```

> **Note**
>
> Jika Anda menggunakan AWS IAM Identity Center untuk otentikasi, pastikan untuk juga menambahkan `AWSSDK.SSO` dan`AWSSDK.SSOOIDC`.

4. Ganti kode `Program.cs` dengan kode berikut.

```
using AWS.Messaging;
using Handler;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
```

```
var builder = Host.CreateDefaultBuilder(args);

builder.ConfigureServices(services =>
{
    // Register the AWS Message Processing Framework for .NET.
    services.AddAWSMessageBus(builder =>
    {
        // Check for input SQS URL.
        // The SQS URL should be passed as a command line argument or set in the
 Debug launch profile.
        if ((args.Length == 1) && (args[0].Contains("https://sqs.")))
        {
            // Register you'll poll the following queue.
            builder.AddSQSPoller(args[0]);

            // And that messages of type "greetingMessage" should be:
            // 1. Deserialized as GreetingMessage objects.
            // 2. Which are then passed to GreetingMessageHandler.
            builder.AddMessageHandler<GreetingMessageHandler,
 GreetingMessage>("greetingMessage");

        }
        // You can add additional message handlers here, using different message
 types.
    });
});

var host = builder.Build();
await host.RunAsync();

namespace Handler
{
    /// <summary>
    /// This class represents the message contents.
    /// </summary>
    public class GreetingMessage
    {
        public string? SenderName { get; set; }
        public string? Greeting { get; set; }
    }

    /// <summary>
    /// This handler is invoked each time you receive the message.
```

```
    /// </summary>
    public class GreetingMessageHandler : IMessageHandler<GreetingMessage>
    {
        public Task<MessageProcessStatus> HandleAsync(
            MessageEnvelope<GreetingMessage> messageEnvelope,
            CancellationToken token = default)
        {
            Console.WriteLine(
                $"Received message {messageEnvelope.Message.Greeting} from
 {messageEnvelope.Message.SenderName}");
            return Task.FromResult(MessageProcessStatus.Success());
        }
    }
}
```

5. Jalankan perintah berikut. Ini memulai poller yang berjalan lama.

```
dotnet run <queue URL created earlier>
```

Tak lama setelah startup aplikasi akan menerima pesan yang dikirim di bagian pertama tutorial ini dan log pesan berikut:

```
Received message {greeting} from {senderName}
```

6. Tekan `Ctrl+C` untuk menghentikan poller.

Pembersihan

Gunakan salah satu perintah berikut untuk AWS CLI atau Alat AWS untuk PowerShell untuk menghapus antrian.

AWS CLI

```
aws sqs delete-queue --queue-url "<queue URL created earlier>"
```

Alat AWS untuk PowerShell

```
Remove-SQSQueue -QueueUrl "<queue URL created earlier>"
```

Publikasikan pesan dengan AWS Message Processing Framework untuk.NET

AWS Message Processing Framework untuk.NET mendukung penerbitan satu atau beberapa jenis pesan, memproses satu atau beberapa jenis pesan, atau melakukan keduanya dalam aplikasi yang sama.

Kode berikut menunjukkan konfigurasi untuk aplikasi yang menerbitkan jenis pesan yang berbeda ke AWS layanan yang berbeda.

```
var builder = WebApplication.CreateBuilder(args);

// Register the AWS Message Processing Framework for .NET
builder.Services.AddAWSMessageBus(builder =>
{
    // Register that you'll send messages of type ChatMessage to an existing queue
    builder.AddSQSPublisher<ChatMessage>("https://sqs.us-
west-2.amazonaws.com/012345678910/MyAppProd");

    // Register that you'll publish messages of type OrderInfo to an existing SNS topic
    builder.AddSNSPublisher<OrderInfo>("arn:aws:sns:us-west-2:012345678910:MyAppProd");

    // Register that you'll publish messages of type FoodItem to an existing
 EventBridge bus
    builder.AddEventBridgePublisher<FoodItem>("arn:aws:events:us-
west-2:012345678910:event-bus/default");
});
```

Setelah Anda mendaftarkan kerangka kerja selama startup, masukkan generik `IMessagePublisher` ke dalam kode Anda. Panggil `PublishAsync` metodenya untuk mempublikasikan salah satu jenis pesan yang dikonfigurasi di atas. Penerbit generik akan menentukan tujuan untuk mengarahkan pesan berdasarkan jenisnya.

Dalam contoh berikut, pengontrol ASP.NET MVC menerima `ChatMessage` pesan dan `OrderInfo` peristiwa dari pengguna, dan kemudian menerbitkannya ke Amazon SQS dan Amazon SNS masing-masing. Kedua jenis pesan dapat dipublikasikan menggunakan penerbit generik yang telah dikonfigurasi di atas.

```
[ApiController]
[Route("[controller]")]
public class PublisherController : ControllerBase
{
    private readonly IMessagePublisher _messagePublisher;
```

```
    public PublisherController(IMessagePublisher messagePublisher)
    {
        _messagePublisher = messagePublisher;
    }

    [HttpPost("chatmessage", Name = "Chat Message")]
    public async Task<IActionResult> PublishChatMessage([FromBody] ChatMessage message)
    {
        // Perform business and validation logic on the ChatMessage here.
        if (message == null)
        {
            return BadRequest("A chat message was not submitted. Unable to forward to
 the message queue.");
        }
        if (string.IsNullOrEmpty(message.MessageDescription))
        {
            return BadRequest("The MessageDescription cannot be null or empty.");
        }

        // Send the ChatMessage to SQS, using the generic publisher.
        await _messagePublisher.PublishAsync(message);

        return Ok();
    }

    [HttpPost("order", Name = "Order")]
    public async Task<IActionResult> PublishOrder([FromBody] OrderInfo message)
    {
        if (message == null)
        {
            return BadRequest("An order was not submitted.");
        }

        // Publish the OrderInfo to SNS, using the generic publisher.
        await _messagePublisher.PublishAsync(message);

        return Ok();
    }
}
```

Untuk merutekan pesan ke logika penanganan yang sesuai, kerangka kerja menggunakan metadata yang disebut pengenal tipe pesan. Secara default, ini adalah nama lengkap dari jenis .NET pesan,

termasuk nama rakitannya. Jika Anda mengirim dan menangani pesan, mekanisme ini berfungsi dengan baik jika Anda membagikan definisi objek pesan Anda di seluruh proyek. Namun, jika pesan didefinisikan ulang di ruang nama yang berbeda, atau jika Anda bertukar pesan dengan kerangka kerja atau bahasa pemrograman lain, Anda mungkin perlu mengganti pengenal jenis pesan.

```
var builder = Host.CreateDefaultBuilder(args);

builder.ConfigureServices(services =>
{
    // Register the AWS Message Processing Framework for .NET
    services.AddAWSMessageBus(builder =>
    {
        // Register that you'll publish messages of type GreetingMessage to an existing
 queue
        builder.AddSQSPublisher<GreetingMessage>("https://sqs.us-
west-2.amazonaws.com/012345678910/MyAppProd", "greetingMessage");
    });
});
```

Penerbit khusus layanan

Contoh yang ditunjukkan di atas menggunakan generik`IMessagePublisher`, yang dapat mempublikasikan ke AWS layanan yang didukung berdasarkan jenis pesan yang dikonfigurasi. Kerangka kerja ini juga menyediakan penerbit khusus layanan untuk Amazon SQS, Amazon SNS, dan Amazon. EventBridge Penayang khusus ini mengekspos opsi yang hanya berlaku untuk layanan itu, dan dapat disuntikkan menggunakan jenis`ISQSPublisher`,, `ISNSPublisher` dan. `IEventBridgePublisher`

Misalnya, saat mengirim pesan ke antrean SQS FIFO, Anda harus mengatur ID grup pesan yang sesuai. Kode berikut menunjukkan `ChatMessage` contoh lagi, tetapi sekarang menggunakan `ISQSPublisher` untuk mengatur opsi khusus SQS.

```
public class PublisherController : ControllerBase
{
    private readonly ISQSPublisher _sqsPublisher;

    public PublisherController(ISQSPublisher sqsPublisher)
    {
        _sqsPublisher = sqsPublisher;
    }
```

```
    [HttpPost("chatmessage", Name = "Chat Message")]
    public async Task<IActionResult> PublishChatMessage([FromBody] ChatMessage message)
    {
        // Perform business and validation logic on the ChatMessage here
        if (message == null)
        {
            return BadRequest("A chat message was not submitted. Unable to forward to
 the message queue.");
        }
        if (string.IsNullOrEmpty(message.MessageDescription))
        {
            return BadRequest("The MessageDescription cannot be null or empty.");
        }

        // Send the ChatMessage to SQS using the injected ISQSPublisher, with SQS-
specific options
        await _sqsPublisher.SendAsync(message, new SQSOptions
        {
            DelaySeconds = <delay-in-seconds>,
            MessageAttributes = <message-attributes>,
            MessageDeduplicationId = <message-deduplication-id>,
            MessageGroupId = <message-group-id>
        });

        return Ok();
    }
}
```

Hal yang sama dapat dilakukan untuk SNS dan EventBridge, menggunakan `ISNSPublisher` dan `IEventBridgePublisher` masing-masing.

```
await _snsPublisher.PublishAsync(message, new SNSOptions
{
    Subject = <subject>,
    MessageAttributes = <message-attributes>,
    MessageDeduplicationId = <message-deduplication-id>,
    MessageGroupId = <message-group-id>
});
```

```
await _eventBridgePublisher.PublishAsync(message, new EventBridgeOptions
{
    DetailType = <detail-type>,
    Resources = <resources>,
```

```
    Source = <source>,
    Time = <time>,
    TraceHeader = <trace-header>
});
```

Secara default, pesan dari jenis tertentu dikirim ke tujuan yang dikonfigurasi sebelumnya. Namun, Anda dapat mengganti tujuan untuk satu pesan menggunakan penayang khusus pesan. Anda juga dapat mengganti AWS SDK untuk .NET klien yang mendasari yang digunakan untuk mempublikasikan pesan, yang dapat berguna dalam aplikasi multi-penyewa di mana Anda perlu mengubah peran atau kredensialnya, tergantung pada tujuan.

```
await _sqsPublisher.SendAsync(message, new SQSOptions
{
    OverrideClient = <override IAmazonSQS client>,
    QueueUrl = <override queue URL>
});
```

Mengkonsumsi pesan dengan AWS Message Processing Framework untuk.NET

Kerangka Pemrosesan AWS Pesan untuk.NET memungkinkan Anda untuk menggunakan pesan yang telah diterbitkan dengan menggunakan kerangka kerja atau salah satu layanan pesan. Pesan dapat dikonsumsi dengan berbagai cara, beberapa di antaranya dijelaskan di bawah ini.

Penangan Pesan

Untuk menggunakan pesan, terapkan penangan pesan menggunakan `IMessageHandler` antarmuka untuk setiap jenis pesan yang ingin Anda proses. Pemetaan antara jenis pesan dan penangan pesan dikonfigurasi dalam startup proyek.

```
await Host.CreateDefaultBuilder(args)
    .ConfigureServices(services =>
    {
        // Register the AWS Message Processing Framework for .NET
        services.AddAWSMessageBus(builder =>
        {
            // Register an SQS Queue that the framework will poll for messages.
            // NOTE: The URL given below is an example. Use the appropriate URL for
 your SQS Queue.
            builder.AddSQSPoller("https://sqs.us-west-2.amazonaws.com/012345678910/
MyAppProd");
```

```
            // Register all IMessageHandler implementations with the message type they
 should process.
            // Here messages that match our ChatMessage .NET type will be handled by
 our ChatMessageHandler
            builder.AddMessageHandler<ChatMessageHandler, ChatMessage>();
        });
    })
    .Build()
    .RunAsync();
```

Kode berikut menunjukkan contoh handler pesan untuk `ChatMessage` pesan.

```
public class ChatMessageHandler : IMessageHandler<ChatMessage>
{
    public Task<MessageProcessStatus> HandleAsync(MessageEnvelope<ChatMessage>
 messageEnvelope, CancellationToken token = default)
    {
        // Add business and validation logic here.
        if (messageEnvelope == null)
        {
            return Task.FromResult(MessageProcessStatus.Failed());
        }

        if (messageEnvelope.Message == null)
        {
            return Task.FromResult(MessageProcessStatus.Failed());
        }

        ChatMessage message = messageEnvelope.Message;

        Console.WriteLine($"Message Description: {message.MessageDescription}");

        // Return success so the framework will delete the message from the queue.
        return Task.FromResult(MessageProcessStatus.Success());
    }
}
```

Bagian luar `MessageEnvelope` berisi metadata yang digunakan oleh kerangka kerja. `message`Properti adalah jenis pesan (dalam hal ini`ChatMessage`).

Anda dapat kembali `MessageProcessStatus.Success()` untuk menunjukkan bahwa pesan telah diproses dengan sukses dan kerangka kerja akan menghapus pesan dari antrian Amazon SQS. Saat

kembali`MessageProcessStatus.Failed()`, pesan akan tetap berada dalam antrian di mana ia dapat diproses lagi atau dipindahkan ke antrian huruf mati, jika dikonfigurasi.

Menangani Pesan dalam Proses yang Berjalan Lama

Anda dapat menelepon `AddSQSPoller` dengan URL antrian SQS untuk memulai jangka panjang `BackgroundService`yang akan terus melakukan polling antrian dan memproses pesan.

```
await Host.CreateDefaultBuilder(args)
    .ConfigureServices(services =>
    {
        // Register the AWS Message Processing Framework for .NET
        services.AddAWSMessageBus(builder =>
        {
            // Register an SQS Queue that the framework will poll for messages.
            // NOTE: The URL given below is an example. Use the appropriate URL for
 your SQS Queue.
            builder.AddSQSPoller("https://sqs.us-west-2.amazonaws.com/012345678910/
MyAppProd", options =>
            {
                // The maximum number of messages from this queue that the framework
 will process concurrently on this client.
                options.MaxNumberOfConcurrentMessages = 10;

                // The duration each call to SQS will wait for new messages.
                options.WaitTimeSeconds = 20;
            });

            // Register all IMessageHandler implementations with the message type they
 should process.
            builder.AddMessageHandler<ChatMessageHandler, ChatMessage>();
        });
    })
    .Build()
    .RunAsync();
```

Mengkonfigurasi SQS Message Poller

Poller pesan SQS dapat dikonfigurasi oleh `SQSMessagePollerOptions` saat memanggil. `AddSQSPoller`

- `MaxNumberOfConcurrentMessages`- Jumlah maksimum pesan dari antrian untuk diproses secara bersamaan. Nilai default adalah 10.

- `WaitTimeSeconds`- Durasi (dalam detik) di mana panggilan `ReceiveMessage` SQS menunggu pesan tiba dalam antrian sebelum kembali. Jika pesan tersedia, panggilan akan kembali lebih cepat dari`WaitTimeSeconds`. Nilai defaultnya adalah 20.

Penanganan Batas Waktu Visibilitas Pesan

Pesan SQS memiliki periode batas waktu visibilitas. Ketika satu konsumen mulai menangani pesan yang diberikan, itu tetap dalam antrian tetapi disembunyikan dari konsumen lain untuk menghindari pemrosesan lebih dari sekali. Jika pesan tidak ditangani dan dihapus sebelum terlihat lagi, konsumen lain mungkin mencoba menangani pesan yang sama.

Framework akan melacak dan mencoba memperpanjang batas waktu visibilitas untuk pesan yang saat ini ditangani. Anda dapat mengonfigurasi perilaku ini pada `SQSMessagePollerOptions` saat meneleponAddSQSPoller.

- `VisibilityTimeout`- Durasi dalam detik yang menerima pesan disembunyikan dari permintaan pengambilan berikutnya. Nilai default-nya adalah 30.
- `VisibilityTimeoutExtensionThreshold`- Ketika batas waktu visibilitas pesan dalam beberapa detik setelah kedaluwarsa, kerangka kerja akan memperpanjang batas waktu visibilitas (beberapa detik lagi). `VisibilityTimeout` Nilai bawaannya adalah 5.
- `VisibilityTimeoutExtensionHeartbeatInterval`- Seberapa sering dalam hitungan detik kerangka kerja akan memeriksa pesan yang dalam hitungan `VisibilityTimeoutExtensionThreshold` detik setelah kedaluwarsa, dan kemudian memperpanjang batas waktu visibilitasnya. Nilai default adalah 1.

Dalam contoh berikut, framework akan memeriksa setiap 1 detik untuk pesan yang masih ditangani. Untuk pesan-pesan tersebut dalam waktu 5 detik setelah terlihat lagi, kerangka kerja akan secara otomatis memperpanjang batas waktu visibilitas setiap pesan dengan 30 detik lagi.

```
// NOTE: The URL given below is an example. Use the appropriate URL for your SQS Queue.
builder.AddSQSPoller("https://sqs.us-west-2.amazonaws.com/012345678910/MyAppProd",
 options =>
{
    options.VisibilityTimeout = 30;
    options.VisibilityTimeoutExtensionThreshold = 5;
    VisibilityTimeoutExtensionHeartbeatInterval = 1;
});
```

Menangani pesan dalam AWS Lambda fungsi

Anda dapat menggunakan AWS Message Processing Framework untuk.NET dengan integrasi SQS dengan Lambda. Ini disediakan oleh `AWS.Messaging.Lambda` paket. Lihat README-nya untuk memulai.

Menggunakan FIFO dengan AWS Message Processing Framework untuk.NET

Untuk kasus penggunaan di mana pengurutan pesan dan deduplikasi pesan sangat penting, Kerangka Pemrosesan AWS Pesan untuk.NET mendukung antrian Amazon SQS first-in-first-out (FIFO) dan topik Amazon SNS.

Publikasi

Saat memublikasikan pesan ke antrian atau topik FIFO, Anda harus menyetel ID grup pesan, yang menentukan grup yang menjadi milik pesan tersebut. Pesan dalam grup diproses secara berurutan. Anda dapat mengatur ini pada penerbit pesan khusus SQS dan khusus SNS.

```
await _sqsPublisher.PublishAsync(message, new SQSOptions
{
    MessageDeduplicationId = <message-deduplication-id>,
    MessageGroupId = <message-group-id>
});
```

Berlangganan

Saat menangani pesan dari antrian FIFO, kerangka kerja menangani pesan dalam grup pesan tertentu sesuai urutan penerimaannya untuk setiap `ReceiveMessages` panggilan. Kerangka kerja memasuki mode operasi ini secara otomatis ketika dikonfigurasi dengan antrian yang diakhiri. `.fifo`

```
await Host.CreateDefaultBuilder(args)
    .ConfigureServices(services =>
    {
        // Register the AWS Message Processing Framework for .NET.
        services.AddAWSMessageBus(builder =>
        {
            // Because this is a FIFO queue, the framework automatically handles these
 messages in order.
            builder.AddSQSPoller("https://sqs.us-west-2.amazonaws.com/012345678910/
MPF.fifo");
            builder.AddMessageHandler<OrderMessageHandler, OrderMessage>();
        });
```

```
    })
    .Build()
    .RunAsync();
```

Logging dan Buka Telemetri untuk Kerangka Pemrosesan AWS Pesan untuk.NET

Kerangka Pemrosesan AWS Pesan untuk.NET diinstrumentasi OpenTelemetry untuk mencatat jejak untuk setiap pesan yang diterbitkan atau ditangani oleh kerangka kerja. Ini disediakan oleh `AWS.Messaging.Telemetry.OpenTelemetry`paket. Lihat README-nya untuk memulai.

> **Note**
>
> Untuk informasi keamanan yang terkait dengan pencatatan, lihatKeamanan untuk Kerangka Pemrosesan AWS Pesan untuk.NET.

Kustomisasi Kerangka Pemrosesan AWS Pesan untuk.NET

AWS Message Processing Framework untuk.NET membangun, mengirim, dan menangani pesan dalam tiga “lapisan” yang berbeda:

1. Pada lapisan terluar, kerangka kerja membangun permintaan AWS-native atau respons khusus untuk layanan. Dengan Amazon SQS misalnya, ia membangun `SendMessage`permintaan, dan bekerja dengan `Message`objek yang ditentukan oleh layanan.
2. Di dalam permintaan dan respons SQS, framework menetapkan `MessageBody` elemen (atau `Message` untuk Amazon SNS `Detail` atau EventBridge Amazon) ke format JSON. CloudEvent Ini berisi metadata yang ditetapkan oleh kerangka kerja yang dapat diakses pada `MessageEnvelope` objek saat menangani pesan.
3. Pada lapisan terdalam, `data` atribut di dalam objek CloudEvent JSON berisi serialisasi JSON dari objek.NET yang dikirim atau diterima sebagai pesan.

```
{
    "id":"b02f156b-0f02-48cf-ae54-4fbbe05cffba",
    "source":"/aws/messaging",
    "specversion":"1.0",
    "type":"Publisher.Models.ChatMessage",
    "time":"2023-11-21T16:36:02.8957126+00:00",
    "data":"<the ChatMessage object serialized as JSON>"
}
```

Anda dapat menyesuaikan bagaimana amplop pesan dikonfigurasi dan membaca:

- `"id"`secara unik mengidentifikasi pesan. Secara default ini disetel ke GUID baru, tetapi ini dapat diganti dengan mengimplementasikan milik Anda sendiri `IMessageIdGenerator` dan menyuntikkannya ke dalam wadah DI.
- `"type"`mengontrol bagaimana pesan dirutekan ke penangan. Secara default ini menggunakan nama lengkap tipe.NET yang sesuai dengan pesan. Anda dapat mengganti ini melalui `messageTypeIdentifier` parameter saat memetakan jenis pesan ke tujuan melalui`AddSQSPublisher`,`AddSNSPublisher`, atau. `AddEventBridgePublisher`
- `"source"`menunjukkan sistem atau server mana yang mengirim pesan.
  - Ini akan menjadi nama fungsi jika menerbitkan dari AWS Lambda, nama cluster dan tugas ARN jika di Amazon ECS, ID instance jika di Amazon EC2, jika tidak nilai fallback dari. `/aws/messaging`
  - Anda dapat mengganti ini melalui `AddMessageSource` atau `AddMessageSourceSuffix` di. `MessageBusBuilder`
- `"time"`diatur ke arus DateTime di UTC. Ini dapat diganti dengan mengimplementasikan milik Anda sendiri `IDateTimeHandler` dan menyuntikkannya ke dalam wadah DI.
- `"data"`berisi representasi JSON dari objek.NET yang dikirim atau diterima sebagai pesan:
  - `ConfigureSerializationOptions`on `MessageBusBuilder` memungkinkan Anda untuk mengonfigurasi `System.Text.Json.JsonSerializerOptions`yang akan digunakan saat membuat serial dan deserialisasi pesan.
  - Untuk menyuntikkan atribut tambahan atau mengubah amplop pesan setelah kerangka kerja membangunnya, Anda dapat menerapkan `ISerializationCallback` dan mendaftarkannya melalui on. `AddSerializationCallback` `MessageBusBuilder`

Keamanan untuk Kerangka Pemrosesan AWS Pesan untuk.NET

AWS Message Processing Framework untuk.NET bergantung pada AWS SDK untuk .NET untuk berkomunikasi dengan. AWS Untuk informasi lebih lanjut tentang keamanan di AWS SDK untuk .NET, lihatKeamanan untuk AWS Produk atau Layanan ini.

Untuk tujuan keamanan, framework tidak mencatat pesan data yang dikirim oleh pengguna. Jika Anda ingin mengaktifkan fungsi ini untuk tujuan debugging, Anda perlu memanggil `EnableDataMessageLogging()` Bus Pesan sebagai berikut:

```
builder.Services.AddAWSMessageBus(bus =>
```

```
{
    builder.EnableDataMessageLogging();
});
```

Jika Anda menemukan potensi masalah keamanan, lihat kebijakan keamanan untuk melaporkan informasi.

Integrasi AWS dengan .NET Aspire di AWS SDK untuk .NET

.NET Aspire adalah cara baru untuk membangun aplikasi cloud-ready. Secara khusus, ini menyediakan orkestrasi untuk lingkungan lokal di mana untuk menjalankan, menghubungkan, dan men-debug komponen aplikasi terdistribusi. Untuk meningkatkan loop dev internal untuk aplikasi siap cloud, integrasi dengan .NET Aspire telah dibuat untuk menghubungkan aplikasi.NET Anda ke AWS sumber daya. Integrasi ini tersedia melalui paket NuGet Aspire.Hosting.aws.

Integrasi.NET Aspire berikut tersedia:

- Kemampuan untuk menyediakan AWS sumber daya Anda melalui CloudFormation. Integrasi ini digunakan dalam AppHost proyek .NET Aspire.

  Untuk informasi lebih lanjut, lihat posting blog Mengintegrasikan AWS dengan.NET Aspire.
- Menginstal, mengonfigurasi, dan menghubungkan AWS SDK untuk .NET ke Amazon DynamoDB lokal. Integrasi ini digunakan dalam AppHost proyek .NET Aspire.

  Untuk informasi lebih lanjut, lihat posting blog Mengintegrasikan AWS dengan.NET Aspire.
- Aktifkan lingkungan pengembangan lokal untuk AWS Lambdafungsi. Integrasi ini digunakan dalam AppHost proyek .NET Aspire.

  Untuk informasi lebih lanjut, lihat posting blog Membangun dan Debugging aplikasi.NET Lambda dengan.NET Aspire (Bagian 1) dan Membangun dan Debugging. NET Lambda aplikasi.NET dengan .NET Aspire (Bagian 2).

> **Note**
>
> Ini adalah dokumentasi prarilis untuk fitur dalam rilis pratinjau. Dokumentasi ini dapat berubah.

Karena fitur ini dalam pratinjau, Anda harus ikut serta untuk fitur pratinjau. Untuk informasi tambahan tentang fitur pratinjau ini dan cara ikut serta, lihat masalah pelacak pengembangan di. GitHub

Informasi tambahan

Untuk informasi tambahan dan detail tentang cara menggunakan integrasi yang tersedia di Aspire.Hosting.aws, lihat sumber daya berikut.

- Posting blog Mengintegrasikan AWS dengan .NET Aspire.
- Posting blog Membangun dan Debugging aplikasi.NET Lambda dengan.NET Aspire (Bagian 1) dan Membangun dan Debugging aplikasi.NET Lambda dengan.NET Aspire (Bagian 2).
- integrations-on-dotnet-aspireRepo -for-aws aktif. GitHub
- README terperinci untuk paket NuGet Aspire.Hosting.aws.

Support untuk AWS layanan dan konfigurasi lainnya

AWS Layanan AWS SDK untuk .NET pendukung selain yang dijelaskan di bagian sebelumnya. Untuk informasi tentang APIs untuk semua layanan yang didukung, lihat Referensi AWS SDK untuk .NET API.

Selain ruang nama untuk AWS layanan individu, AWS SDK untuk .NET juga menyediakan yang berikut: APIs

Bidang	Deskripsi	Sumber daya
AWS Support	Akses terprogram ke kasus AWS Support dan fitur Trusted Advisor.	Lihat Amazon. AWSSupport dan Amazon. AWSSupport.Model.
Umum	Kelas pembantu dan enumerasi.	Lihat Amazon dan Amazon.Util.

SDK untuk .NET (v4) contoh kode

Contoh kode dalam topik ini menunjukkan cara menggunakan AWS SDK untuk .NET (v4) dengan AWS.

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

Tindakan merupakan 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

- Contoh Aurora menggunakan SDK untuk .NET (v4)
- Contoh Auto Scaling menggunakan SDK untuk .NET (v4)
- Contoh Amazon Bedrock menggunakan SDK untuk .NET (v4)
- Contoh Amazon Bedrock Runtime menggunakan SDK untuk .NET (v4)
- CloudFormation contoh menggunakan SDK untuk .NET (v4)
- CloudWatch contoh menggunakan SDK untuk .NET (v4)
- CloudWatch Log contoh menggunakan SDK untuk .NET (v4)
- Contoh Penyedia Identitas Amazon Cognito menggunakan SDK untuk .NET (v4)
- AWS Control Tower contoh menggunakan SDK untuk .NET (v4)
- Contoh SDK untuk .NET DynamoDB menggunakan (v4)
- Contoh Amazon EC2 menggunakan SDK untuk .NET (v4)
- Contoh Amazon ECS menggunakan SDK untuk .NET (v4)
- AWS IoT contoh menggunakan SDK untuk .NET (v4)
- AWS IoT data contoh menggunakan SDK untuk .NET (v4)
- Contoh Amazon Redshift menggunakan SDK untuk .NET (v4)

- Contoh Amazon S3 menggunakan SDK untuk .NET (v4)

Contoh Aurora menggunakan SDK untuk .NET (v4)

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK untuk .NET (v4) dengan Aurora.

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

Tindakan merupakan 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

- Memulai
- Hal-hal mendasar
- Tindakan

Memulai

Halo Aurora

Contoh kode berikut menunjukkan bagaimana memulai menggunakan Aurora.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
using Amazon.RDS;
```

```
using Amazon.RDS.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

namespace AuroraActions;

public static class HelloAurora
{
    static async Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
 the
        // Amazon Relational Database Service (Amazon RDS).
        // Use your AWS profile name, or leave it blank to use the default profile.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonRDS>()
            ).Build();

        // Now the client is available for injection. Fetching it directly here for
 example purposes only.
        var rdsClient = host.Services.GetRequiredService<IAmazonRDS>();

        // You can use await and any of the async methods to get a response.
        var response = await rdsClient.DescribeDBClustersAsync(new
 DescribeDBClustersRequest { IncludeShared = true });
        Console.WriteLine($"Hello Amazon RDS Aurora! Let's list some clusters in
 this account:");
        if (response.DBClusters == null)
        {
            Console.WriteLine($"\tNo clusters found.");
        }
        else
        {
            foreach (var cluster in response.DBClusters)
            {
                Console.WriteLine(
                    $"\tCluster: database: {cluster.DatabaseName} identifier:
 {cluster.DBClusterIdentifier}.");
            }
        }
    }
}
```

- Untuk detail API, lihat Menjelaskan DBClusters di Referensi AWS SDK untuk .NET API.

Hal-hal mendasar

Pelajari dasar-dasarnya

Contoh kode berikut ini menunjukkan cara untuk melakukan:

- 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 .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkap dan pelajari cara menyiapkan dan menjalankan di Repositori Contoh Kode AWS.

Jalankan skenario interaktif di penggugah/prompt perintah.

```
using Amazon.RDS;
using Amazon.RDS.Model;
using AuroraActions;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;

namespace AuroraScenario;

/// <summary>
/// Scenario for Amazon Aurora examples.
/// </summary>
```

```
public class AuroraScenario
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.

    This .NET example performs the following tasks:
    1.  Return a list of the available DB engine families for Aurora MySql using the
 DescribeDBEngineVersionsAsync method.
    2.  Select an engine family and create a custom DB cluster parameter group using
 the CreateDBClusterParameterGroupAsync method.
    3.  Get the parameter group using the DescribeDBClusterParameterGroupsAsync
 method.
    4.  Get some parameters in the group using the DescribeDBClusterParametersAsync
 method.
    5.  Parse and display some parameters in the group.
    6.  Modify the auto_increment_offset and auto_increment_increment parameters
        using the ModifyDBClusterParameterGroupAsync method.
    7.  Get and display the updated parameters using the
 DescribeDBClusterParametersAsync method with a source of "user".
    8.  Get a list of allowed engine versions using the
 DescribeDBEngineVersionsAsync method.
    9.  Create an Aurora DB cluster that contains a MySql database and uses the
 parameter group.
        using the CreateDBClusterAsync method.
    10. Wait for the DB cluster to be ready using the DescribeDBClustersAsync
 method.
    11. Display and select from a list of instance classes available for the
 selected engine and version
        using the paginated DescribeOrderableDBInstanceOptions method.
    12. Create a database instance in the cluster using the CreateDBInstanceAsync
 method.
    13. Wait for the DB instance to be ready using the DescribeDBInstances method.
    14. Display the connection endpoint string for the new DB cluster.
    15. Create a snapshot of the DB cluster using the CreateDBClusterSnapshotAsync
 method.
    16. Wait for DB snapshot to be ready using the DescribeDBClusterSnapshotsAsync
 method.
    17. Delete the DB instance using the DeleteDBInstanceAsync method.
    18. Delete the DB cluster using the DeleteDBClusterAsync method.
    19. Wait for DB cluster to be deleted using the DescribeDBClustersAsync methods.
    20. Delete the cluster parameter group using the
 DeleteDBClusterParameterGroupAsync.
    */
```

```
   private static readonly string sepBar = new('-', 80);
   private static AuroraWrapper auroraWrapper = null!;
   private static ILogger logger = null!;
   private static readonly string engine = "aurora-mysql";
   static async Task Main(string[] args)
   {
       // Set up dependency injection for the Amazon Relational Database Service
(Amazon RDS).
       using var host = Host.CreateDefaultBuilder(args)
           .ConfigureLogging(logging =>
               logging.AddFilter("System", LogLevel.Debug)
                   .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                   .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
           .ConfigureServices((_, services) =>
               services.AddAWSService<IAmazonRDS>()
                   .AddTransient<AuroraWrapper>()
           )
           .Build();

       logger = LoggerFactory.Create(builder =>
       {
           builder.AddConsole();
       }).CreateLogger<AuroraScenario>();

       auroraWrapper = host.Services.GetRequiredService<AuroraWrapper>();

       Console.WriteLine(sepBar);
       Console.WriteLine(
           "Welcome to the Amazon Aurora: get started with DB clusters example.");
       Console.WriteLine(sepBar);

       DBClusterParameterGroup parameterGroup = null!;
       DBCluster? newCluster = null;
       DBInstance? newInstance = null;

       try
       {
           var parameterGroupFamily = await ChooseParameterGroupFamilyAsync();

           parameterGroup = await
CreateDBParameterGroupAsync(parameterGroupFamily);
```

```
            var parameters = await
DescribeParametersInGroupAsync(parameterGroup.DBClusterParameterGroupName,
                new List<string> { "auto_increment_offset",
"auto_increment_increment" });

            await ModifyParametersAsync(parameterGroup.DBClusterParameterGroupName,
parameters);

            await
DescribeUserSourceParameters(parameterGroup.DBClusterParameterGroupName);

            var engineVersionChoice = await
ChooseDBEngineVersionAsync(parameterGroupFamily);

            var newClusterIdentifier = "Example-Cluster-" + DateTime.Now.Ticks;

            newCluster = await CreateNewCluster
            (
                parameterGroup,
                engine,
                engineVersionChoice.EngineVersion,
                newClusterIdentifier
            );

            var instanceClassChoice = await ChooseDBInstanceClass(engine,
engineVersionChoice.EngineVersion);

            var newInstanceIdentifier = "Example-Instance-" + DateTime.Now.Ticks;

            newInstance = await CreateNewInstance(
                newClusterIdentifier,
                engine,
                engineVersionChoice.EngineVersion,
                instanceClassChoice.DBInstanceClass,
                newInstanceIdentifier
            );

            DisplayConnectionString(newCluster!);
            await CreateSnapshot(newCluster!);
            await CleanupResources(newInstance, newCluster, parameterGroup);

            Console.WriteLine("Scenario complete.");
            Console.WriteLine(sepBar);
        }
```

```
        catch (Exception ex)
        {
            await CleanupResources(newInstance, newCluster, parameterGroup);
            logger.LogError(ex, "There was a problem executing the scenario.");
        }
    }

    /// <summary>
    /// Choose the Aurora DB parameter group family from a list of available
options.
    /// </summary>
    /// <returns>The selected parameter group family.</returns>
    public static async Task<string> ChooseParameterGroupFamilyAsync()
    {
        Console.WriteLine(sepBar);
        // 1. Get a list of available engines.
        var engines = await
auroraWrapper.DescribeDBEngineVersionsForEngineAsync(engine);

        Console.WriteLine($"1. The following is a list of available DB parameter
group families for engine {engine}:");

        var parameterGroupFamilies =
            engines.GroupBy(e => e.DBParameterGroupFamily).ToList();
        for (var i = 1; i <= parameterGroupFamilies.Count; i++)
        {
            var parameterGroupFamily = parameterGroupFamilies[i - 1];
            // List the available parameter group families.
            Console.WriteLine(
                $"\t{i}. Family: {parameterGroupFamily.Key}");
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > parameterGroupFamilies.Count)
        {
            Console.WriteLine("2. Select an available DB parameter group family by
entering a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }
        var parameterGroupFamilyChoice = parameterGroupFamilies[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return parameterGroupFamilyChoice.Key;
    }
```

```
    /// <summary>
    /// Create and get information on a DB parameter group.
    /// </summary>
    /// <param name="dbParameterGroupFamily">The DBParameterGroupFamily for the new
 DB parameter group.</param>
    /// <returns>The new DBParameterGroup.</returns>
    public static async Task<DBClusterParameterGroup>
 CreateDBParameterGroupAsync(string dbParameterGroupFamily)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"2. Create new DB parameter group with family
 {dbParameterGroupFamily}:");

        var parameterGroup = await
 auroraWrapper.CreateCustomClusterParameterGroupAsync(
            dbParameterGroupFamily,
            "ExampleParameterGroup-" + DateTime.Now.Ticks,
            "New example parameter group");

        var groupInfo =
            await
 auroraWrapper.DescribeCustomDBClusterParameterGroupAsync(parameterGroup.DBClusterParameterG

        Console.WriteLine(
            $"3. New DB parameter group created: \n\t{groupInfo?.Description}, \n
\tARN {groupInfo?.DBClusterParameterGroupName}");
        Console.WriteLine(sepBar);
        return parameterGroup;
    }

    /// <summary>
    /// Get and describe parameters from a DBParameterGroup.
    /// </summary>
    /// <param name="parameterGroupName">The name of the DBParameterGroup.</param>
    /// <param name="parameterNames">Optional specific names of parameters to
 describe.</param>
    /// <returns>The list of requested parameters.</returns>
    public static async Task<List<Parameter>> DescribeParametersInGroupAsync(string
 parameterGroupName, List<string>? parameterNames = null)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("4. Get some parameters from the group.");
        Console.WriteLine(sepBar);
```

```
        var parameters =
            await
auroraWrapper.DescribeDBClusterParametersInGroupAsync(parameterGroupName);

        var matchingParameters =
            parameters.Where(p => parameterNames == null ||
parameterNames.Contains(p.ParameterName)).ToList();

        Console.WriteLine("5. Parameter information:");
        matchingParameters.ForEach(p =>
            Console.WriteLine(
                $"\n\tParameter: {p.ParameterName}." +
                $"\n\tDescription: {p.Description}." +
                $"\n\tAllowed Values: {p.AllowedValues}." +
                $"\n\tValue: {p.ParameterValue}."));

        Console.WriteLine(sepBar);

        return matchingParameters;
    }

    /// <summary>
    /// Modify a parameter from a DBParameterGroup.
    /// </summary>
    /// <param name="parameterGroupName">Name of the DBParameterGroup.</param>
    /// <param name="parameters">The parameters to modify.</param>
    /// <returns>Async task.</returns>
    public static async Task ModifyParametersAsync(string parameterGroupName,
List<Parameter> parameters)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("6. Modify some parameters in the group.");

        await auroraWrapper.ModifyIntegerParametersInGroupAsync(parameterGroupName,
parameters);

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Describe the user source parameters in the group.
    /// </summary>
    /// <param name="parameterGroupName">The name of the DBParameterGroup.</param>
```

```
    /// <returns>Async task.</returns>
    public static async Task DescribeUserSourceParameters(string parameterGroupName)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("7. Describe updated user source parameters in the
group.");

        var parameters =
            await
auroraWrapper.DescribeDBClusterParametersInGroupAsync(parameterGroupName, "user");

        parameters.ForEach(p =>
            Console.WriteLine(
                $"\n\tParameter: {p.ParameterName}." +
                $"\n\tDescription: {p.Description}." +
                $"\n\tAllowed Values: {p.AllowedValues}." +
                $"\n\tValue: {p.ParameterValue}."));

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Choose a DB engine version.
    /// </summary>
    /// <param name="dbParameterGroupFamily">DB parameter group family for engine
choice.</param>
    /// <returns>The selected engine version.</returns>
    public static async Task<DBEngineVersion> ChooseDBEngineVersionAsync(string
dbParameterGroupFamily)
    {
        Console.WriteLine(sepBar);
        // Get a list of allowed engines.
        var allowedEngines =
            await auroraWrapper.DescribeDBEngineVersionsForEngineAsync(engine,
dbParameterGroupFamily);

        Console.WriteLine($"Available DB engine versions for parameter group family
{dbParameterGroupFamily}:");
        int i = 1;
        foreach (var version in allowedEngines)
        {
            Console.WriteLine(
                $"\t{i}. {version.DBEngineVersionDescription}.");
            i++;
```

```
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > allowedEngines.Count)
        {
            Console.WriteLine("8. Select an available DB engine version by entering
 a number from the list above:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var engineChoice = allowedEngines[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return engineChoice;
    }

    /// <summary>
    /// Create a new RDS DB cluster.
    /// </summary>
    /// <param name="parameterGroup">Parameter group to use for the DB cluster.</
param>
    /// <param name="engineName">Engine to use for the DB cluster.</param>
    /// <param name="engineVersion">Engine version to use for the DB cluster.</
param>
    /// <param name="clusterIdentifier">Cluster identifier to use for the DB
 cluster.</param>
    /// <returns>The new DB cluster.</returns>
    public static async Task<DBCluster?> CreateNewCluster(DBClusterParameterGroup
 parameterGroup,
        string engineName, string engineVersion, string clusterIdentifier)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"9. Create a new DB cluster with identifier
 {clusterIdentifier}.");

        DBCluster newCluster;
        var clusters = await auroraWrapper.DescribeDBClustersPagedAsync();
        var isClusterCreated = clusters.Any(i => i.DBClusterIdentifier ==
 clusterIdentifier);

        if (isClusterCreated)
        {
            Console.WriteLine("Cluster already created.");
```

```
            newCluster = clusters.First(i => i.DBClusterIdentifier ==
 clusterIdentifier);
        }
        else
        {
            Console.WriteLine("Enter an admin username:");
            var username = Console.ReadLine();

            Console.WriteLine("Enter an admin password:");
            var password = Console.ReadLine();

            newCluster = await auroraWrapper.CreateDBClusterWithAdminAsync(
                "ExampleDatabase",
                clusterIdentifier,
                parameterGroup.DBClusterParameterGroupName,
                engineName,
                engineVersion,
                username!,
                password!
            );

            Console.WriteLine("10. Waiting for DB cluster to be ready...");
            while (newCluster.Status != "available")
            {
                Console.Write(".");
                Thread.Sleep(5000);
                clusters = await
 auroraWrapper.DescribeDBClustersPagedAsync(clusterIdentifier);
                newCluster = clusters.First();
            }
        }

        Console.WriteLine(sepBar);
        return newCluster;
    }

    /// <summary>
    /// Choose a DB instance class for a particular engine and engine version.
    /// </summary>
    /// <param name="engine">DB engine for DB instance choice.</param>
    /// <param name="engineVersion">DB engine version for DB instance choice.</
param>
    /// <returns>The selected orderable DB instance option.</returns>
```

```
    public static async Task<OrderableDBInstanceOption> ChooseDBInstanceClass(string
 engine, string engineVersion)
    {
        Console.WriteLine(sepBar);
        // Get a list of allowed DB instance classes.
        var allowedInstances =
            await auroraWrapper.DescribeOrderableDBInstanceOptionsPagedAsync(engine,
 engineVersion);

        Console.WriteLine($"Available DB instance classes for engine {engine} and
 version {engineVersion}:");
        int i = 1;

        foreach (var instance in allowedInstances)
        {
            Console.WriteLine(
                $"\t{i}. Instance class: {instance.DBInstanceClass} (storage type
 {instance.StorageType})");
            i++;
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > allowedInstances.Count)
        {
            Console.WriteLine("11. Select an available DB instance class by entering
 a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var instanceChoice = allowedInstances[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return instanceChoice;
    }

    /// <summary>
    /// Create a new DB instance.
    /// </summary>
    /// <param name="engineName">Engine to use for the DB instance.</param>
    /// <param name="engineVersion">Engine version to use for the DB instance.</
param>
    /// <param name="instanceClass">Instance class to use for the DB instance.</
param>
```

```
   /// <param name="instanceIdentifier">Instance identifier to use for the DB
instance.</param>
   /// <returns>The new DB instance.</returns>
   public static async Task<DBInstance?> CreateNewInstance(
       string clusterIdentifier,
       string engineName,
       string engineVersion,
       string instanceClass,
       string instanceIdentifier)
   {
       Console.WriteLine(sepBar);
       Console.WriteLine($"12. Create a new DB instance with identifier
{instanceIdentifier}.");
       bool isInstanceReady = false;
       DBInstance newInstance;
       var instances = await auroraWrapper.DescribeDBInstancesPagedAsync();
       isInstanceReady = instances.FirstOrDefault(i =>
           i.DBInstanceIdentifier == instanceIdentifier)?.DBInstanceStatus ==
"available";

       if (isInstanceReady)
       {
           Console.WriteLine("Instance already created.");
           newInstance = instances.First(i => i.DBInstanceIdentifier ==
instanceIdentifier);
       }
       else
       {
           newInstance = await auroraWrapper.CreateDBInstanceInClusterAsync(
               clusterIdentifier,
               instanceIdentifier,
               engineName,
               engineVersion,
               instanceClass
           );

           Console.WriteLine("13. Waiting for DB instance to be ready...");
           while (!isInstanceReady)
           {
               Console.Write(".");
               Thread.Sleep(5000);
               instances = await
auroraWrapper.DescribeDBInstancesPagedAsync(instanceIdentifier);
```

```
                isInstanceReady = instances.FirstOrDefault()?.DBInstanceStatus ==
 "available";
                newInstance = instances.First();
            }
        }

        Console.WriteLine(sepBar);
        return newInstance;
    }

    /// <summary>
    /// Display a connection string for an Amazon RDS DB cluster.
    /// </summary>
    /// <param name="cluster">The DB cluster to use to get a connection string.</
param>
    public static void DisplayConnectionString(DBCluster cluster)
    {
        Console.WriteLine(sepBar);
        // Display the connection string.
        Console.WriteLine("14. New DB cluster connection string: ");
        Console.WriteLine(
            $"\n{engine} -h {cluster.Endpoint} -P {cluster.Port} "
            + $"-u {cluster.MasterUsername} -p [YOUR PASSWORD]\n");

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Create a snapshot from an Amazon RDS DB cluster.
    /// </summary>
    /// <param name="cluster">DB cluster to use when creating a snapshot.</param>
    /// <returns>The snapshot object.</returns>
    public static async Task<DBClusterSnapshot> CreateSnapshot(DBCluster cluster)
    {
        Console.WriteLine(sepBar);
        // Create a snapshot.
        Console.WriteLine($"15. Creating snapshot from DB cluster
 {cluster.DBClusterIdentifier}.");
        var snapshot = await auroraWrapper.CreateClusterSnapshotByIdentifierAsync(
            cluster.DBClusterIdentifier,
            "ExampleSnapshot-" + DateTime.Now.Ticks);

        // Wait for the snapshot to be available.
        bool isSnapshotReady = false;
```

```
        Console.WriteLine($"16. Waiting for snapshot to be ready...");
        while (!isSnapshotReady)
        {
            Console.Write(".");
            Thread.Sleep(5000);
            var snapshots =
                await
auroraWrapper.DescribeDBClusterSnapshotsByIdentifierAsync(cluster.DBClusterIdentifier);
            isSnapshotReady = snapshots.FirstOrDefault()?.Status == "available";
            snapshot = snapshots.First();
        }

        Console.WriteLine(
            $"Snapshot {snapshot.DBClusterSnapshotIdentifier} status is
{snapshot.Status}.");
        Console.WriteLine(sepBar);
        return snapshot;
    }

    /// <summary>
    /// Clean up resources from the scenario.
    /// </summary>
    /// <param name="newInstance">The instance to clean up.</param>
    /// <param name="newCluster">The cluster to clean up.</param>
    /// <param name="parameterGroup">The parameter group to clean up.</param>
    /// <returns>Async Task.</returns>
    private static async Task CleanupResources(
        DBInstance? newInstance,
        DBCluster? newCluster,
        DBClusterParameterGroup? parameterGroup)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Clean up resources.");

        if (newInstance is not null && GetYesNoResponse($"\tClean up instance
{newInstance.DBInstanceIdentifier}? (y/n)"))
        {
            // Delete the DB instance.
            Console.WriteLine($"17. Deleting the DB instance
{newInstance.DBInstanceIdentifier}.");
            await
auroraWrapper.DeleteDBInstanceByIdentifierAsync(newInstance.DBInstanceIdentifier);
        }
```

```
        if (newCluster is not null && GetYesNoResponse($"\tClean up cluster
{newCluster.DBClusterIdentifier}? (y/n)"))
        {
            // Delete the DB cluster.
            Console.WriteLine($"18. Deleting the DB cluster
{newCluster.DBClusterIdentifier}.");
            await
auroraWrapper.DeleteDBClusterByIdentifierAsync(newCluster.DBClusterIdentifier);

            // Wait for the DB cluster to delete.
            Console.WriteLine($"19. Waiting for the DB cluster to delete...");
            bool isClusterDeleted = false;

            while (!isClusterDeleted)
            {
                Console.Write(".");
                Thread.Sleep(5000);
                var cluster = await auroraWrapper.DescribeDBClustersPagedAsync();
                isClusterDeleted = cluster.All(i => i.DBClusterIdentifier !=
newCluster.DBClusterIdentifier);
            }

            Console.WriteLine("DB cluster deleted.");
        }

        if (parameterGroup is not null && GetYesNoResponse($"\tClean up parameter
group? (y/n)"))
        {
            Console.WriteLine($"20. Deleting the DB parameter group
{parameterGroup.DBClusterParameterGroupName}.");
            await
auroraWrapper.DeleteClusterParameterGroupByNameAsync(parameterGroup.DBClusterParameterGroup
            Console.WriteLine("Parameter group deleted.");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</param>
    /// <returns>True if the user responds with a yes.</returns>
```

```
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null &&
                       ynResponse.Equals("y",
                           StringComparison.InvariantCultureIgnoreCase);
        return response;
    }
```

Metode pembungkus yang dipanggil oleh skenario untuk mengelola tindakan Aurora.

```
using Amazon.RDS;
using Amazon.RDS.Model;

namespace AuroraActions;

/// <summary>
/// Wrapper for the Amazon Aurora cluster client operations.
/// </summary>
public class AuroraWrapper
{
    private readonly IAmazonRDS _amazonRDS;
    public AuroraWrapper(IAmazonRDS amazonRDS)
    {
        _amazonRDS = amazonRDS;
    }

    /// <summary>
    /// Get a list of DB engine versions for a particular DB engine.
    /// </summary>
    /// <param name="engine">The name of the engine.</param>
    /// <param name="parameterGroupFamily">Optional parameter group family name.</
param>
    /// <returns>A list of DBEngineVersions.</returns>
    public async Task<List<DBEngineVersion>>
 DescribeDBEngineVersionsForEngineAsync(string engine,
        string? parameterGroupFamily = null)
    {
        var response = await _amazonRDS.DescribeDBEngineVersionsAsync(
            new DescribeDBEngineVersionsRequest()
```

```
            {
                Engine = engine,
                DBParameterGroupFamily = parameterGroupFamily
            });
        return response.DBEngineVersions;
    }

    /// <summary>
    /// Create a custom cluster parameter group.
    /// </summary>
    /// <param name="parameterGroupFamily">The family of the parameter group.</
param>
    /// <param name="groupName">The name for the new parameter group.</param>
    /// <param name="description">A description for the new parameter group.</param>
    /// <returns>The new parameter group object.</returns>
    public async Task<DBClusterParameterGroup>
 CreateCustomClusterParameterGroupAsync(
        string parameterGroupFamily,
        string groupName,
        string description)
    {
        var request = new CreateDBClusterParameterGroupRequest
        {
            DBParameterGroupFamily = parameterGroupFamily,
            DBClusterParameterGroupName = groupName,
            Description = description,
        };

        var response = await _amazonRDS.CreateDBClusterParameterGroupAsync(request);
        return response.DBClusterParameterGroup;
    }

    /// <summary>
    /// Describe the cluster parameters in a parameter group.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <param name="source">The optional name of the source filter.</param>
    /// <returns>The collection of parameters.</returns>
    public async Task<List<Parameter>>
 DescribeDBClusterParametersInGroupAsync(string groupName, string? source = null)
    {
        var paramList = new List<Parameter>();

        DescribeDBClusterParametersResponse response;
```

```
        var request = new DescribeDBClusterParametersRequest
        {
            DBClusterParameterGroupName = groupName,
            Source = source,
        };

        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClusterParametersAsync(request);
            paramList.AddRange(response.Parameters);

            request.Marker = response.Marker;
        }
        while (response.Marker is not null);

        return paramList;
    }

    /// <summary>
    /// Get the description of a DB cluster parameter group by name.
    /// </summary>
    /// <param name="name">The name of the DB parameter group to describe.</param>
    /// <returns>The parameter group description.</returns>
    public async Task<DBClusterParameterGroup?>
DescribeCustomDBClusterParameterGroupAsync(string name)
    {
        var response = await _amazonRDS.DescribeDBClusterParameterGroupsAsync(
            new DescribeDBClusterParameterGroupsRequest()
            {
                DBClusterParameterGroupName = name
            });
        return response.DBClusterParameterGroups.FirstOrDefault();
    }

    /// <summary>
    /// Modify the specified integer parameters with new values from user input.
    /// </summary>
    /// <param name="groupName">The group name for the parameters.</param>
    /// <param name="parameters">The list of integer parameters to modify.</param>
    /// <param name="newValue">Optional int value to set for parameters.</param>
    /// <returns>The name of the group that was modified.</returns>
    public async Task<string> ModifyIntegerParametersInGroupAsync(string groupName,
List<Parameter> parameters, int newValue = 0)
```

```
    {
        foreach (var p in parameters)
        {
            if (p.IsModifiable.GetValueOrDefault() && p.DataType == "integer")
            {
                while (newValue == 0)
                {
                    Console.WriteLine(
                        $"Enter a new value for {p.ParameterName} from the allowed
values {p.AllowedValues} ");

                    var choice = Console.ReadLine();
                    int.TryParse(choice, out newValue);
                }

                p.ParameterValue = newValue.ToString();
            }
        }

        var request = new ModifyDBClusterParameterGroupRequest
        {
            Parameters = parameters,
            DBClusterParameterGroupName = groupName,
        };

        var result = await _amazonRDS.ModifyDBClusterParameterGroupAsync(request);
        return result.DBClusterParameterGroupName;
    }


    /// <summary>
    /// Get a list of orderable DB instance options for a specific
    /// engine and engine version.
    /// </summary>
    /// <param name="engine">Name of the engine.</param>
    /// <param name="engineVersion">Version of the engine.</param>
    /// <returns>List of OrderableDBInstanceOptions.</returns>
    public async Task<List<OrderableDBInstanceOption>>
DescribeOrderableDBInstanceOptionsPagedAsync(string engine, string engineVersion)
    {
        // Use a paginator to get a list of DB instance options.
        var results = new List<OrderableDBInstanceOption>();
        var paginateInstanceOptions =
_amazonRDS.Paginators.DescribeOrderableDBInstanceOptions(
```

```
            new DescribeOrderableDBInstanceOptionsRequest()
            {
                Engine = engine,
                EngineVersion = engineVersion,
            });
        // Get the entire list using the paginator.
        await foreach (var instanceOptions in
paginateInstanceOptions.OrderableDBInstanceOptions)
        {
            results.Add(instanceOptions);
        }
        return results;
    }

    /// <summary>
    /// Delete a particular parameter group by name.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteClusterParameterGroupByNameAsync(string groupName)
    {
        var request = new DeleteDBClusterParameterGroupRequest
        {
            DBClusterParameterGroupName = groupName,
        };

        var response = await _amazonRDS.DeleteDBClusterParameterGroupAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Create a new cluster and database.
    /// </summary>
    /// <param name="dbName">The name of the new database.</param>
    /// <param name="clusterIdentifier">The identifier of the cluster.</param>
    /// <param name="parameterGroupName">The name of the parameter group.</param>
    /// <param name="dbEngine">The engine to use for the new cluster.</param>
    /// <param name="dbEngineVersion">The version of the engine to use.</param>
    /// <param name="adminName">The admin username.</param>
    /// <param name="adminPassword">The primary admin password.</param>
    /// <returns>The cluster object.</returns>
    public async Task<DBCluster> CreateDBClusterWithAdminAsync(
        string dbName,
        string clusterIdentifier,
```

```
        string parameterGroupName,
        string dbEngine,
        string dbEngineVersion,
        string adminName,
        string adminPassword)
    {
        var request = new CreateDBClusterRequest
        {
            DatabaseName = dbName,
            DBClusterIdentifier = clusterIdentifier,
            DBClusterParameterGroupName = parameterGroupName,
            Engine = dbEngine,
            EngineVersion = dbEngineVersion,
            MasterUsername = adminName,
            MasterUserPassword = adminPassword,
        };

        var response = await _amazonRDS.CreateDBClusterAsync(request);
        return response.DBCluster;
    }

    /// <summary>
    /// Returns a list of DB instances.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB
instance.</param>
    /// <returns>List of DB instances.</returns>
    public async Task<List<DBInstance>> DescribeDBInstancesPagedAsync(string?
dbInstanceIdentifier = null)
    {
        var results = new List<DBInstance>();
        var instancesPaginator = _amazonRDS.Paginators.DescribeDBInstances(
            new DescribeDBInstancesRequest
            {
                DBInstanceIdentifier = dbInstanceIdentifier
            });
        // Get the entire list using the paginator.
        await foreach (var instances in instancesPaginator.DBInstances)
        {
            results.Add(instances);
        }
        return results;
    }
```

```
    /// <summary>
    /// Returns a list of DB clusters.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB cluster.</
param>
    /// <returns>List of DB clusters.</returns>
    public async Task<List<DBCluster>> DescribeDBClustersPagedAsync(string?
 dbClusterIdentifier = null)
    {
        var results = new List<DBCluster>();

        DescribeDBClustersResponse response;
        DescribeDBClustersRequest request = new DescribeDBClustersRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClustersAsync(request);
            if (response.DBClusters != null)
            {
                results.AddRange(response.DBClusters);
            }
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }

    /// <summary>
    /// Create an Amazon Relational Database Service (Amazon RDS) DB instance
    /// with a particular set of properties. Use the action DescribeDBInstancesAsync
    /// to determine when the DB instance is ready to use.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="dbEngine">The engine for the DB instance.</param>
    /// <param name="dbEngineVersion">Version for the DB instance.</param>
    /// <param name="instanceClass">Class for the DB instance.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> CreateDBInstanceInClusterAsync(
        string dbClusterIdentifier,
        string dbInstanceIdentifier,
```

```
        string dbEngine,
        string dbEngineVersion,
        string instanceClass)
    {
        // When creating the instance within a cluster, do not specify the name or
size.
        var response = await _amazonRDS.CreateDBInstanceAsync(
            new CreateDBInstanceRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBInstanceIdentifier = dbInstanceIdentifier,
                Engine = dbEngine,
                EngineVersion = dbEngineVersion,
                DBInstanceClass = instanceClass
            });

        return response.DBInstance;
    }

    /// <summary>
    /// Create a snapshot of a cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
    /// <returns>DB snapshot object.</returns>
    public async Task<DBClusterSnapshot>
CreateClusterSnapshotByIdentifierAsync(string dbClusterIdentifier, string
snapshotIdentifier)
    {
        var response = await _amazonRDS.CreateDBClusterSnapshotAsync(
            new CreateDBClusterSnapshotRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBClusterSnapshotIdentifier = snapshotIdentifier,
            });

        return response.DBClusterSnapshot;
    }

    /// <summary>
    /// Return a list of DB snapshots for a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>List of DB snapshots.</returns>
```

```
   public async Task<List<DBClusterSnapshot>>
DescribeDBClusterSnapshotsByIdentifierAsync(string dbClusterIdentifier)
   {
       var results = new List<DBClusterSnapshot>();

       DescribeDBClusterSnapshotsResponse response;
       DescribeDBClusterSnapshotsRequest request = new
DescribeDBClusterSnapshotsRequest
       {
           DBClusterIdentifier = dbClusterIdentifier
       };
       // Get the full list if there are multiple pages.
       do
       {
           response = await _amazonRDS.DescribeDBClusterSnapshotsAsync(request);
           results.AddRange(response.DBClusterSnapshots);
           request.Marker = response.Marker;
       }
       while (response.Marker is not null);
       return results;
   }

   /// <summary>
   /// Delete a particular DB cluster.
   /// </summary>
   /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
   /// <returns>DB cluster object.</returns>
   public async Task<DBCluster> DeleteDBClusterByIdentifierAsync(string
dbClusterIdentifier)
   {
       var response = await _amazonRDS.DeleteDBClusterAsync(
           new DeleteDBClusterRequest()
           {
               DBClusterIdentifier = dbClusterIdentifier,
               SkipFinalSnapshot = true
           });

       return response.DBCluster;
   }

   /// <summary>
   /// Delete a particular DB instance.
   /// </summary>
   /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
```

```
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> DeleteDBInstanceByIdentifierAsync(string
 dbInstanceIdentifier)
    {
        var response = await _amazonRDS.DeleteDBInstanceAsync(
            new DeleteDBInstanceRequest()
            {
                DBInstanceIdentifier = dbInstanceIdentifier,
                SkipFinalSnapshot = true,
                DeleteAutomatedBackups = true
            });

        return response.DBInstance;
    }
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk .NET .
  - Buat DBCluster
  - Buat DBCluster ParameterGroup
  - Buat DBCluster Snapshot
  - Buat DBInstance
  - Hapus DBCluster
  - Hapus DBCluster ParameterGroup
  - Hapus DBInstance
  - Jelaskan DBCluster ParameterGroups
  - Jelaskan DBCluster Parameter
  - Jelaskan DBCluster Snapshots
  - Jelaskan DBClusters
  - Jelaskan DBEngine Versi
  - Jelaskan DBInstances
  - DescribeOrderableDBInstancePilihan
  - Memodifikasi DBCluster ParameterGroup

Tindakan

`CreateDBCluster`

Contoh kode berikut menunjukkan cara menggunakan`CreateDBCluster`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
/// <summary>
/// Create a new cluster and database.
/// </summary>
/// <param name="dbName">The name of the new database.</param>
/// <param name="clusterIdentifier">The identifier of the cluster.</param>
/// <param name="parameterGroupName">The name of the parameter group.</param>
/// <param name="dbEngine">The engine to use for the new cluster.</param>
/// <param name="dbEngineVersion">The version of the engine to use.</param>
/// <param name="adminName">The admin username.</param>
/// <param name="adminPassword">The primary admin password.</param>
/// <returns>The cluster object.</returns>
public async Task<DBCluster> CreateDBClusterWithAdminAsync(
    string dbName,
    string clusterIdentifier,
    string parameterGroupName,
    string dbEngine,
    string dbEngineVersion,
    string adminName,
    string adminPassword)
{
    var request = new CreateDBClusterRequest
    {
        DatabaseName = dbName,
        DBClusterIdentifier = clusterIdentifier,
        DBClusterParameterGroupName = parameterGroupName,
        Engine = dbEngine,
        EngineVersion = dbEngineVersion,
        MasterUsername = adminName,
```

```
            MasterUserPassword = adminPassword,
        };

        var response = await _amazonRDS.CreateDBClusterAsync(request);
        return response.DBCluster;
    }
```

- Untuk detail API, lihat Membuat DBCluster di Referensi AWS SDK untuk .NET API.

`CreateDBClusterParameterGroup`

Contoh kode berikut menunjukkan cara menggunakan`CreateDBClusterParameterGroup`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Create a custom cluster parameter group.
    /// </summary>
    /// <param name="parameterGroupFamily">The family of the parameter group.</
param>
    /// <param name="groupName">The name for the new parameter group.</param>
    /// <param name="description">A description for the new parameter group.</param>
    /// <returns>The new parameter group object.</returns>
    public async Task<DBClusterParameterGroup>
 CreateCustomClusterParameterGroupAsync(
        string parameterGroupFamily,
        string groupName,
        string description)
    {
        var request = new CreateDBClusterParameterGroupRequest
        {
            DBParameterGroupFamily = parameterGroupFamily,
            DBClusterParameterGroupName = groupName,
            Description = description,
```

```
        };

        var response = await _amazonRDS.CreateDBClusterParameterGroupAsync(request);
        return response.DBClusterParameterGroup;
    }
```

- Untuk detail API, lihat Membuat DBCluster ParameterGroup di Referensi AWS SDK untuk .NET API.

`CreateDBClusterSnapshot`

Contoh kode berikut menunjukkan cara menggunakan`CreateDBClusterSnapshot`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Create a snapshot of a cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
    /// <returns>DB snapshot object.</returns>
    public async Task<DBClusterSnapshot>
CreateClusterSnapshotByIdentifierAsync(string dbClusterIdentifier, string
snapshotIdentifier)
    {
        var response = await _amazonRDS.CreateDBClusterSnapshotAsync(
            new CreateDBClusterSnapshotRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBClusterSnapshotIdentifier = snapshotIdentifier,
            });

        return response.DBClusterSnapshot;
    }
```

- Untuk detail API, lihat Membuat DBCluster Snapshot di Referensi AWS SDK untuk .NET API.

`CreateDBInstance`

Contoh kode berikut menunjukkan cara menggunakan`CreateDBInstance`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Create an Amazon Relational Database Service (Amazon RDS) DB instance
    /// with a particular set of properties. Use the action DescribeDBInstancesAsync
    /// to determine when the DB instance is ready to use.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="dbEngine">The engine for the DB instance.</param>
    /// <param name="dbEngineVersion">Version for the DB instance.</param>
    /// <param name="instanceClass">Class for the DB instance.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> CreateDBInstanceInClusterAsync(
        string dbClusterIdentifier,
        string dbInstanceIdentifier,
        string dbEngine,
        string dbEngineVersion,
        string instanceClass)
    {
        // When creating the instance within a cluster, do not specify the name or
size.
        var response = await _amazonRDS.CreateDBInstanceAsync(
            new CreateDBInstanceRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBInstanceIdentifier = dbInstanceIdentifier,
```

```
                Engine = dbEngine,
                EngineVersion = dbEngineVersion,
                DBInstanceClass = instanceClass
            });

        return response.DBInstance;
    }
```

- Untuk detail API, lihat Membuat DBInstance di Referensi AWS SDK untuk .NET API.

`DeleteDBCluster`

Contoh kode berikut menunjukkan cara menggunakan`DeleteDBCluster`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Delete a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>DB cluster object.</returns>
    public async Task<DBCluster> DeleteDBClusterByIdentifierAsync(string
dbClusterIdentifier)
    {
        var response = await _amazonRDS.DeleteDBClusterAsync(
            new DeleteDBClusterRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                SkipFinalSnapshot = true
            });

        return response.DBCluster;
    }
```

- Untuk detail API, lihat Menghapus DBCluster di Referensi AWS SDK untuk .NET API.

`DeleteDBClusterParameterGroup`

Contoh kode berikut menunjukkan cara menggunakan`DeleteDBClusterParameterGroup`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
/// <summary>
/// Delete a particular parameter group by name.
/// </summary>
/// <param name="groupName">The name of the parameter group.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteClusterParameterGroupByNameAsync(string groupName)
{
    var request = new DeleteDBClusterParameterGroupRequest
    {
        DBClusterParameterGroupName = groupName,
    };

    var response = await _amazonRDS.DeleteDBClusterParameterGroupAsync(request);
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Untuk detail API, lihat Menghapus DBCluster ParameterGroup di Referensi AWS SDK untuk .NET API.

`DeleteDBInstance`

Contoh kode berikut menunjukkan cara menggunakan`DeleteDBInstance`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Delete a particular DB instance.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> DeleteDBInstanceByIdentifierAsync(string
dbInstanceIdentifier)
    {
        var response = await _amazonRDS.DeleteDBInstanceAsync(
            new DeleteDBInstanceRequest()
            {
                DBInstanceIdentifier = dbInstanceIdentifier,
                SkipFinalSnapshot = true,
                DeleteAutomatedBackups = true
            });

        return response.DBInstance;
    }
```

- Untuk detail API, lihat Menghapus DBInstance di Referensi AWS SDK untuk .NET API.

`DescribeDBClusterParameterGroups`

Contoh kode berikut menunjukkan cara menggunakan`DescribeDBClusterParameterGroups`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Get the description of a DB cluster parameter group by name.
    /// </summary>
    /// <param name="name">The name of the DB parameter group to describe.</param>
    /// <returns>The parameter group description.</returns>
    public async Task<DBClusterParameterGroup?>
DescribeCustomDBClusterParameterGroupAsync(string name)
    {
        var response = await _amazonRDS.DescribeDBClusterParameterGroupsAsync(
            new DescribeDBClusterParameterGroupsRequest()
            {
                DBClusterParameterGroupName = name
            });
        return response.DBClusterParameterGroups.FirstOrDefault();
    }
```

- Untuk detail API, lihat Menjelaskan DBCluster ParameterGroups di Referensi AWS SDK untuk .NET API.

`DescribeDBClusterParameters`

Contoh kode berikut menunjukkan cara menggunakan`DescribeDBClusterParameters`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Describe the cluster parameters in a parameter group.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <param name="source">The optional name of the source filter.</param>
    /// <returns>The collection of parameters.</returns>
    public async Task<List<Parameter>>
DescribeDBClusterParametersInGroupAsync(string groupName, string? source = null)
    {
```

```
        var paramList = new List<Parameter>();

        DescribeDBClusterParametersResponse response;
        var request = new DescribeDBClusterParametersRequest
        {
            DBClusterParameterGroupName = groupName,
            Source = source,
        };

        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClusterParametersAsync(request);
            paramList.AddRange(response.Parameters);

            request.Marker = response.Marker;
        }
        while (response.Marker is not null);

        return paramList;
    }
```

- Untuk detail API, lihat Menjelaskan DBCluster Parameter di Referensi AWS SDK untuk .NET API.

`DescribeDBClusterSnapshots`

Contoh kode berikut menunjukkan cara menggunakan`DescribeDBClusterSnapshots`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Return a list of DB snapshots for a particular DB cluster.
    /// </summary>
```

```
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>List of DB snapshots.</returns>
    public async Task<List<DBClusterSnapshot>>
DescribeDBClusterSnapshotsByIdentifierAsync(string dbClusterIdentifier)
    {
        var results = new List<DBClusterSnapshot>();

        DescribeDBClusterSnapshotsResponse response;
        DescribeDBClusterSnapshotsRequest request = new
DescribeDBClusterSnapshotsRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClusterSnapshotsAsync(request);
            results.AddRange(response.DBClusterSnapshots);
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }
```

- Untuk detail API, lihat Menjelaskan DBCluster Snapshot di Referensi AWS SDK untuk .NET API.

`DescribeDBClusters`

Contoh kode berikut menunjukkan cara menggunakan`DescribeDBClusters`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
```

```
    /// Returns a list of DB clusters.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB cluster.</
param>
    /// <returns>List of DB clusters.</returns>
    public async Task<List<DBCluster>> DescribeDBClustersPagedAsync(string?
 dbClusterIdentifier = null)
    {
        var results = new List<DBCluster>();

        DescribeDBClustersResponse response;
        DescribeDBClustersRequest request = new DescribeDBClustersRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClustersAsync(request);
            if (response.DBClusters != null)
            {
                results.AddRange(response.DBClusters);
            }
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }
```

- Untuk detail API, lihat Menjelaskan DBClusters di Referensi AWS SDK untuk .NET API.

`DescribeDBEngineVersions`

Contoh kode berikut menunjukkan cara menggunakan`DescribeDBEngineVersions`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Get a list of DB engine versions for a particular DB engine.
    /// </summary>
    /// <param name="engine">The name of the engine.</param>
    /// <param name="parameterGroupFamily">Optional parameter group family name.</
param>
    /// <returns>A list of DBEngineVersions.</returns>
    public async Task<List<DBEngineVersion>>
 DescribeDBEngineVersionsForEngineAsync(string engine,
        string? parameterGroupFamily = null)
    {
        var response = await _amazonRDS.DescribeDBEngineVersionsAsync(
            new DescribeDBEngineVersionsRequest()
            {
                Engine = engine,
                DBParameterGroupFamily = parameterGroupFamily
            });
        return response.DBEngineVersions;
    }
```

- Untuk detail API, lihat Menjelaskan DBEngine Versi dalam Referensi AWS SDK untuk .NET API.

`DescribeDBInstances`

Contoh kode berikut menunjukkan cara menggunakan`DescribeDBInstances`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Returns a list of DB instances.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB
 instance.</param>
```

```
    /// <returns>List of DB instances.</returns>
    public async Task<List<DBInstance>> DescribeDBInstancesPagedAsync(string?
dbInstanceIdentifier = null)
    {
        var results = new List<DBInstance>();
        var instancesPaginator = _amazonRDS.Paginators.DescribeDBInstances(
            new DescribeDBInstancesRequest
            {
                DBInstanceIdentifier = dbInstanceIdentifier
            });
        // Get the entire list using the paginator.
        await foreach (var instances in instancesPaginator.DBInstances)
        {
            results.Add(instances);
        }
        return results;
    }
```

- Untuk detail API, lihat Menjelaskan DBInstances di Referensi AWS SDK untuk .NET API.

`DescribeOrderableDBInstanceOptions`

Contoh kode berikut menunjukkan cara menggunakan`DescribeOrderableDBInstanceOptions`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Get a list of orderable DB instance options for a specific
    /// engine and engine version.
    /// </summary>
    /// <param name="engine">Name of the engine.</param>
    /// <param name="engineVersion">Version of the engine.</param>
    /// <returns>List of OrderableDBInstanceOptions.</returns>
    public async Task<List<OrderableDBInstanceOption>>
DescribeOrderableDBInstanceOptionsPagedAsync(string engine, string engineVersion)
```

```
    {
        // Use a paginator to get a list of DB instance options.
        var results = new List<OrderableDBInstanceOption>();
        var paginateInstanceOptions =
_amazonRDS.Paginators.DescribeOrderableDBInstanceOptions(
            new DescribeOrderableDBInstanceOptionsRequest()
            {
                Engine = engine,
                EngineVersion = engineVersion,
            });
        // Get the entire list using the paginator.
        await foreach (var instanceOptions in
paginateInstanceOptions.OrderableDBInstanceOptions)
        {
            results.Add(instanceOptions);
        }
        return results;
    }
```

- Untuk detail API, lihat DescribeOrderableDBInstanceOpsi di Referensi AWS SDK untuk .NET API.

`ModifyDBClusterParameterGroup`

Contoh kode berikut menunjukkan cara menggunakan`ModifyDBClusterParameterGroup`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Modify the specified integer parameters with new values from user input.
    /// </summary>
    /// <param name="groupName">The group name for the parameters.</param>
    /// <param name="parameters">The list of integer parameters to modify.</param>
    /// <param name="newValue">Optional int value to set for parameters.</param>
```

```
    /// <returns>The name of the group that was modified.</returns>
    public async Task<string> ModifyIntegerParametersInGroupAsync(string groupName,
List<Parameter> parameters, int newValue = 0)
    {
        foreach (var p in parameters)
        {
            if (p.IsModifiable.GetValueOrDefault() && p.DataType == "integer")
            {
                while (newValue == 0)
                {
                    Console.WriteLine(
                        $"Enter a new value for {p.ParameterName} from the allowed
values {p.AllowedValues} ");

                    var choice = Console.ReadLine();
                    int.TryParse(choice, out newValue);
                }

                p.ParameterValue = newValue.ToString();
            }
        }

        var request = new ModifyDBClusterParameterGroupRequest
        {
            Parameters = parameters,
            DBClusterParameterGroupName = groupName,
        };

        var result = await _amazonRDS.ModifyDBClusterParameterGroupAsync(request);
        return result.DBClusterParameterGroupName;
    }
```

- Untuk detail API, lihat Memodifikasi DBCluster ParameterGroup dalam Referensi AWS SDK untuk .NET API.

Contoh Auto Scaling menggunakan SDK untuk .NET (v4)

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK untuk .NET (v4) dengan Auto Scaling.

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

Tindakan merupakan 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

- Memulai
- Hal-hal mendasar
- Tindakan

Memulai

Halo Auto Scaling

Contoh kode berikut menunjukkan cara memulai menggunakan Auto Scaling.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
namespace AutoScalingActions;

using Amazon.AutoScaling;

public class HelloAutoScaling
{
    /// <summary>
    /// Hello Amazon EC2 Auto Scaling. List EC2 Auto Scaling groups.
    /// </summary>
    /// <param name="args"></param>
```

```
    /// <returns>Async Task.</returns>
    static async Task Main(string[] args)
    {
        var client = new AmazonAutoScalingClient();

        Console.WriteLine("Welcome to Amazon EC2 Auto Scaling.");
        Console.WriteLine("Let's get a description of your Auto Scaling groups.");

        var response = await client.DescribeAutoScalingGroupsAsync();

        if (response.AutoScalingGroups == null || response.AutoScalingGroups.Count
 == 0)
        {
            Console.WriteLine("Sorry, you don't have any Amazon EC2 Auto Scaling
 groups.");
            return;
        }
        response.AutoScalingGroups.ForEach(autoScalingGroup =>
        {

 Console.WriteLine($"{autoScalingGroup.AutoScalingGroupName}\t{autoScalingGroup.Availability
        });

    }
}
```

- Untuk detail API, lihat DescribeAutoScalingGroupsdi Referensi AWS SDK untuk .NET API.

Hal-hal mendasar

Pelajari dasar-dasarnya

Contoh kode berikut ini menunjukkan cara untuk melakukan:

- Buat grup Amazon EC2 Auto Scaling 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 .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
global using Amazon.AutoScaling;
global using Amazon.AutoScaling.Model;
global using Amazon.CloudWatch;
global using AutoScalingActions;
global using Microsoft.Extensions.DependencyInjection;
global using Microsoft.Extensions.Hosting;
global using Microsoft.Extensions.Logging;
global using Microsoft.Extensions.Logging.Console;
global using Microsoft.Extensions.Logging.Debug;



using Amazon.EC2;
using Microsoft.Extensions.Configuration;
using Host = Microsoft.Extensions.Hosting.Host;

namespace AutoScalingBasics;

public class AutoScalingBasics
{

    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon EC2 Auto Scaling, Amazon
        // CloudWatch, and Amazon EC2.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
```

```
                    .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonAutoScaling>()
                .AddAWSService<IAmazonCloudWatch>()
                .AddAWSService<IAmazonEC2>()
                .AddTransient<AutoScalingWrapper>()
                .AddTransient<CloudWatchWrapper>()
                .AddTransient<EC2Wrapper>()
                .AddTransient<UIWrapper>()
            )
            .Build();


        var autoScalingWrapper =
host.Services.GetRequiredService<AutoScalingWrapper>();
        var cloudWatchWrapper =
host.Services.GetRequiredService<CloudWatchWrapper>();
        var ec2Wrapper = host.Services.GetRequiredService<EC2Wrapper>();
        var uiWrapper = host.Services.GetRequiredService<UIWrapper>();

        var configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load test settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally load local settings.
            .Build();

        var imageId = configuration["ImageId"];
        var instanceType = configuration["InstanceType"];
        var launchTemplateName = configuration["LaunchTemplateName"];

        launchTemplateName += Guid.NewGuid().ToString();

        // The name of the Auto Scaling group.
        var groupName = configuration["GroupName"];

        uiWrapper.DisplayTitle("Auto Scaling Basics");
        uiWrapper.DisplayAutoScalingBasicsDescription();

        // Create the launch template and save the template Id to use when deleting
the
        // launch template at the end of the application.
```

```
        var launchTemplateId = await ec2Wrapper.CreateLaunchTemplateAsync(imageId!,
instanceType!, launchTemplateName);

        // Confirm that the template was created by asking for a description of it.
        await ec2Wrapper.DescribeLaunchTemplateAsync(launchTemplateName);

        uiWrapper.PressEnter();

        var availabilityZones = await ec2Wrapper.ListAvailabilityZonesAsync();

        Console.WriteLine($"Creating an Auto Scaling group named {groupName}.");
        await autoScalingWrapper.CreateAutoScalingGroupAsync(
            groupName!,
            launchTemplateName,
            availabilityZones[0].ZoneName);

        // Keep checking the details of the new group until its lifecycle state
        // is "InService".
        Console.WriteLine($"Waiting for the Auto Scaling group to be active.");

        List<AutoScalingInstanceDetails> instanceDetails;

        do
        {
            instanceDetails = await
autoScalingWrapper.DescribeAutoScalingInstancesAsync(groupName!);
        }
        while (instanceDetails.Count <= 0);

        Console.WriteLine($"Auto scaling group {groupName} successfully created.");
        Console.WriteLine($"{instanceDetails.Count} instances were created for the
group.");

        // Display the details of the Auto Scaling group.
        instanceDetails.ForEach(detail =>
        {
            Console.WriteLine($"Group name: {detail.AutoScalingGroupName}");
        });

        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Metrics collection");
        Console.WriteLine($"Enable metrics collection for {groupName}");
        await autoScalingWrapper.EnableMetricsCollectionAsync(groupName!);
```

```
        // Show the metrics that are collected for the group.

        // Update the maximum size of the group to three instances.
        Console.WriteLine("--- Update the Auto Scaling group to increase max size to
3 ---");
        int maxSize = 3;
        await autoScalingWrapper.UpdateAutoScalingGroupAsync(groupName!,
launchTemplateName, maxSize);

        Console.WriteLine("--- Describe all Auto Scaling groups to show the current
state of the group ---");
        var groups = await
autoScalingWrapper.DescribeAutoScalingGroupsAsync(groupName!);

        uiWrapper.DisplayGroupDetails(groups!);

        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Describe account limits");
        await autoScalingWrapper.DescribeAccountLimitsAsync();

        uiWrapper.WaitABit(60, "Waiting for the resources to be ready.");

        uiWrapper.DisplayTitle("Set desired capacity");
        int desiredCapacity = 2;
        await autoScalingWrapper.SetDesiredCapacityAsync(groupName!,
desiredCapacity);

        Console.WriteLine("Get the two instance Id values");

        // Empty the group before getting the details again.
        groups.Clear();
        groups = await
autoScalingWrapper.DescribeAutoScalingGroupsAsync(groupName!);
        if (groups.Any())
        {
            foreach (AutoScalingGroup group in groups)
            {
                Console.WriteLine($"The group name is
{group.AutoScalingGroupName}");
                Console.WriteLine($"The group ARN is {group.AutoScalingGroupARN}");
                var instances = group.Instances;
                foreach (Amazon.AutoScaling.Model.Instance instance in instances)
```

```
                {
                    Console.WriteLine($"The instance id is {instance.InstanceId}");
                    Console.WriteLine($"The lifecycle state is
{instance.LifecycleState}");
                }
            }
        }

        uiWrapper.DisplayTitle("Scaling Activities");
        Console.WriteLine("Let's list the scaling activities that have occurred for
the group.");
        var activities = await
autoScalingWrapper.DescribeScalingActivitiesAsync(groupName!);
        if (activities.Any())
        {
            activities.ForEach(activity =>
            {
                Console.WriteLine($"The activity Id is {activity.ActivityId}");
                Console.WriteLine($"The activity details are {activity.Details}");
            });
        }

        // Display the Amazon CloudWatch metrics that have been collected.
        var metrics = await cloudWatchWrapper.GetCloudWatchMetricsAsync(groupName!);
        if (metrics.Any())
        {
            Console.WriteLine($"Metrics collected for {groupName}:");
            metrics.ForEach(metric =>
            {
                Console.Write($"Metric name: {metric.MetricName}\t");
                Console.WriteLine($"Namespace: {metric.Namespace}");
            });
        }

        var dataPoints = await
cloudWatchWrapper.GetMetricStatisticsAsync(groupName!);
        if (dataPoints.Any())
        {
            Console.WriteLine("Details for the metrics collected:");
            dataPoints.ForEach(detail => { Console.WriteLine(detail); });
        }

        // Disable metrics collection.
        Console.WriteLine("Disabling the collection of metrics for {groupName}.");
```

```
        var success = await
autoScalingWrapper.DisableMetricsCollectionAsync(groupName!);

        if (success)
        {
            Console.WriteLine($"Successfully stopped metrics collection for
{groupName}.");
        }
        else
        {
            Console.WriteLine($"Could not stop metrics collection for
{groupName}.");
        }

        // Terminate all instances in the group.
        uiWrapper.DisplayTitle("Terminating Auto Scaling instances");
        Console.WriteLine("Now terminating all instances in the Auto Scaling
group.");

        if (groups is not null)
        {
            groups.ForEach(group =>
            {
                // Only delete instances in the AutoScaling group we created.
                if (group.AutoScalingGroupName == groupName)
                {
                    group.Instances.ForEach(async instance =>
                    {
                        await
autoScalingWrapper.TerminateInstanceInAutoScalingGroupAsync(instance.InstanceId);
                    });
                }
            });
        }

        // After all instances are terminated, delete the group.
        uiWrapper.DisplayTitle("Clean up resources");
        Console.WriteLine("Deleting the Auto Scaling group.");
        await autoScalingWrapper.DeleteAutoScalingGroupAsync(groupName!);

        // Delete the launch template.
        var deletedLaunchTemplateName = await
ec2Wrapper.DeleteLaunchTemplateAsync(launchTemplateId);
```

```
        if (deletedLaunchTemplateName == launchTemplateName)
        {
            Console.WriteLine("Successfully deleted the launch template.");
        }

        Console.WriteLine("The demo is now concluded.");
    }
}


namespace AutoScalingBasics;

/// <summary>
/// A class to provide user interface methods for the EC2 AutoScaling Basics
/// scenario.
/// </summary>
public class UIWrapper
{
    public readonly string SepBar = new('-', Console.WindowWidth);

    /// <summary>
    /// Describe the steps in the EC2 AutoScaling Basics scenario.
    /// </summary>
    public void DisplayAutoScalingBasicsDescription()
    {
        Console.WriteLine("This code example performs the following operations:");
        Console.WriteLine(" 1. Creates an Amazon EC2 launch template.");
        Console.WriteLine(" 2. Creates an Auto Scaling group.");
        Console.WriteLine(" 3. Shows the details of the new Auto Scaling group");
        Console.WriteLine("    to show that only one instance was created.");
        Console.WriteLine(" 4. Enables metrics collection.");
        Console.WriteLine(" 5. Updates the Auto Scaling group to increase the");
        Console.WriteLine("    capacity to three.");
        Console.WriteLine(" 6. Describes Auto Scaling groups again to show the");
        Console.WriteLine("    current state of the group.");
        Console.WriteLine(" 7. Changes the desired capacity of the Auto Scaling");
        Console.WriteLine("    group to use an additional instance.");
        Console.WriteLine(" 8. Shows that there are now instances in the group.");
        Console.WriteLine(" 9. Lists the scaling activities that have occurred for
 the group.");
        Console.WriteLine("10. Displays the Amazon CloudWatch metrics that have");
        Console.WriteLine("    been collected.");
        Console.WriteLine("11. Disables metrics collection.");
```

```
        Console.WriteLine("12. Terminates all instances in the Auto Scaling
 group.");
        Console.WriteLine("13. Deletes the Auto Scaling group.");
        Console.WriteLine("14. Deletes the Amazon EC2 launch template.");
        PressEnter();
    }

    /// <summary>
    /// Display information about the Amazon Ec2 AutoScaling groups passed
    /// in the list of AutoScalingGroup objects.
    /// </summary>
    /// <param name="groups">A list of AutoScalingGroup objects.</param>
    public void DisplayGroupDetails(List<AutoScalingGroup> groups)
    {
        if (groups is null)
            return;

        groups.ForEach(group =>
        {
            Console.WriteLine($"Group name:\t{group.AutoScalingGroupName}");
            Console.WriteLine($"Group created:\t{group.CreatedTime}");
            Console.WriteLine($"Maximum number of instances:\t{group.MaxSize}");
            Console.WriteLine($"Desired number of instances:
\t{group.DesiredCapacity}");
        });
    }

    /// <summary>
    /// Display a message and wait until the user presses enter.
    /// </summary>
    public void PressEnter()
    {
        Console.Write("\nPress <Enter> to continue. ");
        _ = Console.ReadLine();
        Console.WriteLine();
    }

    /// <summary>
    /// Pad a string with spaces to center it on the console display.
    /// </summary>
    /// <param name="strToCenter">The string to be centered.</param>
    /// <returns>The padded string.</returns>
    public string CenterString(string strToCenter)
    {
```

```
        var padAmount = (Console.WindowWidth - strToCenter.Length) / 2;
        var leftPad = new string(' ', padAmount);
        return $"{leftPad}{strToCenter}";
    }

    /// <summary>
    /// Display a line of hyphens, the centered text of the title and another
    /// line of hyphens.
    /// </summary>
    /// <param name="strTitle">The string to be displayed.</param>
    public void DisplayTitle(string strTitle)
    {
        Console.WriteLine(SepBar);
        Console.WriteLine(CenterString(strTitle));
        Console.WriteLine(SepBar);
    }

    /// <summary>
    /// Display a countdown and wait for a number of seconds.
    /// </summary>
    /// <param name="numSeconds">The number of seconds to wait.</param>
    public void WaitABit(int numSeconds, string msg)
    {
        Console.WriteLine(msg);

        // Wait for the requested number of seconds.
        for (int i = numSeconds; i > 0; i--)
        {
            System.Threading.Thread.Sleep(1000);
            Console.Write($"{i}...");
        }

        PressEnter();
    }
}
```

Tentukan fungsi yang dipanggil oleh skenario untuk mengelola template dan metrik peluncuran. Fungsi-fungsi ini membungkus Auto Scaling, Amazon EC2, dan tindakan. CloudWatch

```
namespace AutoScalingActions;
```

```
using Amazon.AutoScaling;
using Amazon.AutoScaling.Model;

/// <summary>
/// A class that includes methods to perform Amazon EC2 Auto Scaling
/// actions.
/// </summary>
public class AutoScalingWrapper
{
    private readonly IAmazonAutoScaling _amazonAutoScaling;

    /// <summary>
    /// Constructor for the AutoScalingWrapper class.
    /// </summary>
    /// <param name="amazonAutoScaling">The injected Amazon EC2 Auto Scaling
 client.</param>
    public AutoScalingWrapper(IAmazonAutoScaling amazonAutoScaling)
    {
        _amazonAutoScaling = amazonAutoScaling;
    }


    /// <summary>
    /// Create a new Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name to use for the new Auto Scaling
    /// group.</param>
    /// <param name="launchTemplateName">The name of the Amazon EC2 Auto Scaling
    /// launch template to use to create instances in the group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> CreateAutoScalingGroupAsync(
        string groupName,
        string launchTemplateName,
        string availabilityZone)
    {
        var templateSpecification = new LaunchTemplateSpecification
        {
            LaunchTemplateName = launchTemplateName,
        };

        var zoneList = new List<string>
            {
                availabilityZone,
```

```
            };

        var request = new CreateAutoScalingGroupRequest
        {
            AutoScalingGroupName = groupName,
            AvailabilityZones = zoneList,
            LaunchTemplate = templateSpecification,
            MaxSize = 6,
            MinSize = 1
        };
        try
        {
            var response = await
_amazonAutoScaling.CreateAutoScalingGroupAsync(request);
            Console.WriteLine($"{groupName} Auto Scaling Group created");
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AlreadyExistsException)
        {
            Console.WriteLine($"{groupName} Auto Scaling Group already exists.");
            return true;
        }
    }



    /// <summary>
    /// Retrieve information about Amazon EC2 Auto Scaling quotas to the
    /// active AWS account.
    /// </summary>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DescribeAccountLimitsAsync()
    {
        var response = await _amazonAutoScaling.DescribeAccountLimitsAsync();
        Console.WriteLine("The maximum number of Auto Scaling groups is " +
response.MaxNumberOfAutoScalingGroups);
        Console.WriteLine("The current number of Auto Scaling groups is " +
response.NumberOfAutoScalingGroups);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }



    /// <summary>
```

```
    /// Retrieve a list of the Amazon EC2 Auto Scaling activities for an
    /// Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of Amazon EC2 Auto Scaling activities.</returns>
    public async Task<List<Activity>> DescribeScalingActivitiesAsync(
        string groupName)
    {
        var activities = new List<Activity>();
        var scalingActivitiesRequest = new DescribeScalingActivitiesRequest
        {
            AutoScalingGroupName = groupName,
            MaxRecords = 10,
        };

        var response = await
 _amazonAutoScaling.DescribeScalingActivitiesAsync(scalingActivitiesRequest);
        if (response.Activities != null)
        {
            activities = response.Activities;
        }
        return activities;
    }



    /// <summary>
    /// Get data about the instances in an Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of Amazon EC2 Auto Scaling details.</returns>
    public async Task<List<AutoScalingInstanceDetails>>
 DescribeAutoScalingInstancesAsync(
        string groupName)
    {
        var groups = await DescribeAutoScalingGroupsAsync(groupName);
        var instanceIds = new List<string>();
        var instanceDetails = new List<AutoScalingInstanceDetails>();
        if (groups != null)
        {
            groups.ForEach(group =>
            {
```

```
                if (group.AutoScalingGroupName == groupName && group.Instances !=
 null)
                {
                    group.Instances.ForEach(instance =>
                    {
                        instanceIds.Add(instance.InstanceId);
                    });
                }
            });

            var scalingGroupsRequest = new DescribeAutoScalingInstancesRequest
            {
                MaxRecords = 10,
                InstanceIds = instanceIds,
            };

            var response =
                await _amazonAutoScaling.DescribeAutoScalingInstancesAsync(
                    scalingGroupsRequest);
            if (response.AutoScalingInstances != null)
            {
                instanceDetails = response.AutoScalingInstances;
            }
        }

        return instanceDetails;
    }



    /// <summary>
    /// Retrieve a list of information about Amazon EC2 Auto Scaling groups.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of Amazon EC2 Auto Scaling groups.</returns>
    public async Task<List<AutoScalingGroup>> DescribeAutoScalingGroupsAsync(
        string groupName)
    {
        var groups = new List<AutoScalingGroup>();
        var groupList = new List<string>
            {
                groupName,
            };
```

```
        var request = new DescribeAutoScalingGroupsRequest
        {
            AutoScalingGroupNames = groupList,
        };

        var response = await
 _amazonAutoScaling.DescribeAutoScalingGroupsAsync(request);
        if (response.AutoScalingGroups != null)
        {
            groups = response.AutoScalingGroups;
        }

        return groups;
    }


    /// <summary>
    /// Delete an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteAutoScalingGroupAsync(
        string groupName)
    {
        var deleteAutoScalingGroupRequest = new DeleteAutoScalingGroupRequest
        {
            AutoScalingGroupName = groupName,
            ForceDelete = true,
        };

        var response = await
 _amazonAutoScaling.DeleteAutoScalingGroupAsync(deleteAutoScalingGroupRequest);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"You successfully deleted {groupName}");
            return true;
        }

        Console.WriteLine($"Couldn't delete {groupName}.");
        return false;
    }
```

```
    /// <summary>
    /// Disable the collection of metric data for an Amazon EC2 Auto Scaling
    /// group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <returns>A Boolean value that indicates the success or failure of
    /// the operation.</returns>
    public async Task<bool> DisableMetricsCollectionAsync(string groupName)
    {
        var request = new DisableMetricsCollectionRequest
        {
            AutoScalingGroupName = groupName,
        };

        var response = await
_amazonAutoScaling.DisableMetricsCollectionAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Enable the collection of metric data for an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> EnableMetricsCollectionAsync(string groupName)
    {
        var listMetrics = new List<string>
            {
                "GroupMaxSize",
            };

        var collectionRequest = new EnableMetricsCollectionRequest
        {
            AutoScalingGroupName = groupName,
            Metrics = listMetrics,
            Granularity = "1Minute",
        };

        var response = await
_amazonAutoScaling.EnableMetricsCollectionAsync(collectionRequest);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

```
    /// <summary>
    /// Set the desired capacity of an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <param name="desiredCapacity">The desired capacity for the Auto
    /// Scaling group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> SetDesiredCapacityAsync(
        string groupName,
        int desiredCapacity)
    {
        var capacityRequest = new SetDesiredCapacityRequest
        {
            AutoScalingGroupName = groupName,
            DesiredCapacity = desiredCapacity,
        };

        var response = await
 _amazonAutoScaling.SetDesiredCapacityAsync(capacityRequest);
        Console.WriteLine($"You have set the DesiredCapacity to
 {desiredCapacity}.");

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Terminate all instances in the Auto Scaling group in preparation for
    /// deleting the group.
    /// </summary>
    /// <param name="instanceId">The instance Id of the instance to terminate.</
param>
    /// <returns>A Boolean value that indicates the success or failure of
    /// the operation.</returns>
    public async Task<bool> TerminateInstanceInAutoScalingGroupAsync(
        string instanceId)
    {
        var request = new TerminateInstanceInAutoScalingGroupRequest
        {
            InstanceId = instanceId,
            ShouldDecrementDesiredCapacity = false,
        };
```

```
        var response = await
 _amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"You have terminated the instance: {instanceId}");
            return true;
        }

        Console.WriteLine($"Could not terminate {instanceId}");
        return false;
    }


    /// <summary>
    /// Update the capacity of an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <param name="launchTemplateName">The name of the EC2 launch template.</
param>
    /// <param name="maxSize">The maximum number of instances that can be
    /// created for the Auto Scaling group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> UpdateAutoScalingGroupAsync(
        string groupName,
        string launchTemplateName,
        int maxSize)
    {
        var templateSpecification = new LaunchTemplateSpecification
        {
            LaunchTemplateName = launchTemplateName,
        };

        var groupRequest = new UpdateAutoScalingGroupRequest
        {
            MaxSize = maxSize,
            AutoScalingGroupName = groupName,
            LaunchTemplate = templateSpecification,
        };

        var response = await
 _amazonAutoScaling.UpdateAutoScalingGroupAsync(groupRequest);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
```

```
        {
            Console.WriteLine($"You successfully updated the Auto Scaling group
 {groupName}.");
            return true;
        }
        else
        {
            return false;
        }
    }

}


namespace AutoScalingActions;

using Amazon.EC2;
using Amazon.EC2.Model;

public class EC2Wrapper
{
    private readonly IAmazonEC2 _amazonEc2;

    /// <summary>
    /// Constructor for the EC2Wrapper class.
    /// </summary>
    /// <param name="amazonEc2">The injected Amazon EC2 client.</param>
    public EC2Wrapper(IAmazonEC2 amazonEc2)
    {
        _amazonEc2 = amazonEc2;
    }

    /// <summary>
    /// Create a new Amazon EC2 launch template.
    /// </summary>
    /// <param name="imageId">The image Id to use for instances launched
    /// using the Amazon EC2 launch template.</param>
    /// <param name="instanceType">The type of EC2 instances to create.</param>
    /// <param name="launchTemplateName">The name of the launch template.</param>
    /// <returns>Returns the TemplateID of the new launch template.</returns>
    public async Task<string> CreateLaunchTemplateAsync(
        string imageId,
        string instanceType,
        string launchTemplateName)
```

```
    {
        var request = new CreateLaunchTemplateRequest
        {
            LaunchTemplateData = new RequestLaunchTemplateData
            {
                ImageId = imageId,
                InstanceType = instanceType,
            },
            LaunchTemplateName = launchTemplateName,
        };

        var response = await _amazonEc2.CreateLaunchTemplateAsync(request);

        return response.LaunchTemplate.LaunchTemplateId;
    }

    /// <summary>
    /// Delete an Amazon EC2 launch template.
    /// </summary>
    /// <param name="launchTemplateId">The TemplateId of the launch template to
    /// delete.</param>
    /// <returns>The name of the EC2 launch template that was deleted.</returns>
    public async Task<string> DeleteLaunchTemplateAsync(string launchTemplateId)
    {
        var request = new DeleteLaunchTemplateRequest
        {
            LaunchTemplateId = launchTemplateId,
        };

        var response = await _amazonEc2.DeleteLaunchTemplateAsync(request);
        return response.LaunchTemplate.LaunchTemplateName;
    }

    /// <summary>
    /// Retrieve information about an EC2 launch template.
    /// </summary>
    /// <param name="launchTemplateName">The name of the EC2 launch template.</
param>
    /// <returns>A Boolean value that indicates the success or failure of
    /// the operation.</returns>
    public async Task<bool> DescribeLaunchTemplateAsync(string launchTemplateName)
    {
        var request = new DescribeLaunchTemplatesRequest
        {
```

```
            LaunchTemplateNames = new List<string> { launchTemplateName, },
        };

        var response = await _amazonEc2.DescribeLaunchTemplatesAsync(request);

        if (response.LaunchTemplates is not null)
        {
            response.LaunchTemplates.ForEach(template =>
            {
                Console.Write($"{template.LaunchTemplateName}\t");
                Console.WriteLine(template.LaunchTemplateId);
            });

            return true;
        }

        return false;
    }

    /// <summary>
    /// Retrieve the availability zones for the current region.
    /// </summary>
    /// <returns>A collection of availability zones.</returns>
    public async Task<List<AvailabilityZone>> ListAvailabilityZonesAsync()
    {
        var response = await _amazonEc2.DescribeAvailabilityZonesAsync(
            new DescribeAvailabilityZonesRequest());

        return response.AvailabilityZones;
    }
}


namespace AutoScalingActions;

using Amazon.CloudWatch;
using Amazon.CloudWatch.Model;

/// <summary>
/// Contains methods to access Amazon CloudWatch metrics for the
/// Amazon EC2 Auto Scaling basics scenario.
/// </summary>
public class CloudWatchWrapper
{
```

```
    private readonly IAmazonCloudWatch _amazonCloudWatch;

    /// <summary>
    /// Constructor for the CloudWatchWrapper.
    /// </summary>
    /// <param name="amazonCloudWatch">The injected CloudWatch client.</param>
    public CloudWatchWrapper(IAmazonCloudWatch amazonCloudWatch)
    {
        _amazonCloudWatch = amazonCloudWatch;
    }

    /// <summary>
    /// Retrieve the metrics information collection for the Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <returns>A list of Metrics collected for the Auto Scaling group.</returns>
    public async Task<List<Metric>> GetCloudWatchMetricsAsync(string groupName)
    {
        var metrics = new List<Metric>();
        var filter = new DimensionFilter
        {
            Name = "AutoScalingGroupName",
            Value = $"{groupName}",
        };

        var request = new ListMetricsRequest
        {
            MetricName = "AutoScalingGroupName",
            Dimensions = new List<DimensionFilter> { filter },
            Namespace = "AWS/AutoScaling",
        };

        var response = await _amazonCloudWatch.ListMetricsAsync(request);
        if (response.Metrics != null)
        {
            metrics = response.Metrics;
        }
        return metrics;
    }

    /// <summary>
    /// Retrieve the metric data collected for an Amazon EC2 Auto Scaling group.
    /// </summary>
```

```
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of data points.</returns>
    public async Task<List<Datapoint>> GetMetricStatisticsAsync(string groupName)
    {
        var dataPoints = new List<Datapoint>();
        var metricDimensions = new List<Dimension>
            {
                new Dimension
                {
                    Name = "AutoScalingGroupName",
                    Value = $"{groupName}",
                },
            };

        // The start time will be yesterday.
        var startTime = DateTime.UtcNow.AddDays(-1);

        var request = new GetMetricStatisticsRequest
        {
            MetricName = "AutoScalingGroupName",
            Dimensions = metricDimensions,
            Namespace = "AWS/AutoScaling",
            Period = 60, // 60 seconds.
            Statistics = new List<string>() { "Minimum" },
            StartTimeUtc = startTime,
            EndTimeUtc = DateTime.UtcNow,
        };

        var response = await _amazonCloudWatch.GetMetricStatisticsAsync(request);
        if (response.Datapoints != null)
        {
            dataPoints = response.Datapoints;
        }

        return dataPoints;
    }

}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk .NET .

- CreateAutoScalingGroup
- DeleteAutoScalingGroup
- DescribeAutoScalingGroups
- DescribeAutoScalingInstances
- DescribeScalingActivities
- DisableMetricsCollection
- EnableMetricsCollection
- SetDesiredCapacity
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

Tindakan

`CreateAutoScalingGroup`

Contoh kode berikut menunjukkan cara menggunakan`CreateAutoScalingGroup`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
/// <summary>
/// Create a new Amazon EC2 Auto Scaling group.
/// </summary>
/// <param name="groupName">The name to use for the new Auto Scaling
/// group.</param>
/// <param name="launchTemplateName">The name of the Amazon EC2 Auto Scaling
/// launch template to use to create instances in the group.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> CreateAutoScalingGroupAsync(
    string groupName,
    string launchTemplateName,
```

```
        string availabilityZone)
    {
        var templateSpecification = new LaunchTemplateSpecification
        {
            LaunchTemplateName = launchTemplateName,
        };

        var zoneList = new List<string>
            {
                availabilityZone,
            };

        var request = new CreateAutoScalingGroupRequest
        {
            AutoScalingGroupName = groupName,
            AvailabilityZones = zoneList,
            LaunchTemplate = templateSpecification,
            MaxSize = 6,
            MinSize = 1
        };
        try
        {
            var response = await
_amazonAutoScaling.CreateAutoScalingGroupAsync(request);
            Console.WriteLine($"{groupName} Auto Scaling Group created");
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AlreadyExistsException)
        {
            Console.WriteLine($"{groupName} Auto Scaling Group already exists.");
            return true;
        }
    }
```

- Untuk detail API, lihat CreateAutoScalingGroupdi Referensi AWS SDK untuk .NET API.

`DescribeAutoScalingGroups`

Contoh kode berikut menunjukkan cara menggunakan`DescribeAutoScalingGroups`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Get data about the instances in an Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of Amazon EC2 Auto Scaling details.</returns>
    public async Task<List<AutoScalingInstanceDetails>>
 DescribeAutoScalingInstancesAsync(
        string groupName)
    {
        var groups = await DescribeAutoScalingGroupsAsync(groupName);
        var instanceIds = new List<string>();
        var instanceDetails = new List<AutoScalingInstanceDetails>();
        if (groups != null)
        {
            groups.ForEach(group =>
            {
                if (group.AutoScalingGroupName == groupName && group.Instances !=
 null)
                {
                    group.Instances.ForEach(instance =>
                    {
                        instanceIds.Add(instance.InstanceId);
                    });
                }
            });

            var scalingGroupsRequest = new DescribeAutoScalingInstancesRequest
            {
                MaxRecords = 10,
                InstanceIds = instanceIds,
            };
```

```
            var response =
                await _amazonAutoScaling.DescribeAutoScalingInstancesAsync(
                    scalingGroupsRequest);
            if (response.AutoScalingInstances != null)
            {
                instanceDetails = response.AutoScalingInstances;
            }
        }

        return instanceDetails;
    }
```

- Untuk detail API, lihat DescribeAutoScalingGroupsdi Referensi AWS SDK untuk .NET API.

`DescribeAutoScalingInstances`

Contoh kode berikut menunjukkan cara menggunakan`DescribeAutoScalingInstances`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Get data about the instances in an Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of Amazon EC2 Auto Scaling details.</returns>
    public async Task<List<AutoScalingInstanceDetails>>
 DescribeAutoScalingInstancesAsync(
        string groupName)
    {
        var groups = await DescribeAutoScalingGroupsAsync(groupName);
        var instanceIds = new List<string>();
```

```
        var instanceDetails = new List<AutoScalingInstanceDetails>();
        if (groups != null)
        {
            groups.ForEach(group =>
            {
                if (group.AutoScalingGroupName == groupName && group.Instances !=
null)
                {
                    group.Instances.ForEach(instance =>
                    {
                        instanceIds.Add(instance.InstanceId);
                    });
                }
            });

            var scalingGroupsRequest = new DescribeAutoScalingInstancesRequest
            {
                MaxRecords = 10,
                InstanceIds = instanceIds,
            };

            var response =
                await _amazonAutoScaling.DescribeAutoScalingInstancesAsync(
                    scalingGroupsRequest);
            if (response.AutoScalingInstances != null)
            {
                instanceDetails = response.AutoScalingInstances;
            }
        }

        return instanceDetails;
    }
```

- Untuk detail API, lihat DescribeAutoScalingInstancesdi Referensi AWS SDK untuk .NET API.

`DescribeScalingActivities`

Contoh kode berikut menunjukkan cara menggunakan`DescribeScalingActivities`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Retrieve a list of the Amazon EC2 Auto Scaling activities for an
    /// Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of Amazon EC2 Auto Scaling activities.</returns>
    public async Task<List<Activity>> DescribeScalingActivitiesAsync(
        string groupName)
    {
        var activities = new List<Activity>();
        var scalingActivitiesRequest = new DescribeScalingActivitiesRequest
        {
            AutoScalingGroupName = groupName,
            MaxRecords = 10,
        };

        var response = await
 _amazonAutoScaling.DescribeScalingActivitiesAsync(scalingActivitiesRequest);
        if (response.Activities != null)
        {
            activities = response.Activities;
        }
        return activities;
    }
```

- Untuk detail API, lihat DescribeScalingActivitiesdi Referensi AWS SDK untuk .NET API.

`DisableMetricsCollection`

Contoh kode berikut menunjukkan cara menggunakan`DisableMetricsCollection`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Disable the collection of metric data for an Amazon EC2 Auto Scaling
    /// group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <returns>A Boolean value that indicates the success or failure of
    /// the operation.</returns>
    public async Task<bool> DisableMetricsCollectionAsync(string groupName)
    {
        var request = new DisableMetricsCollectionRequest
        {
            AutoScalingGroupName = groupName,
        };

        var response = await
_amazonAutoScaling.DisableMetricsCollectionAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- Untuk detail API, lihat DisableMetricsCollectiondi Referensi AWS SDK untuk .NET API.

`EnableMetricsCollection`

Contoh kode berikut menunjukkan cara menggunakan`EnableMetricsCollection`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Enable the collection of metric data for an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> EnableMetricsCollectionAsync(string groupName)
    {
        var listMetrics = new List<string>
            {
                "GroupMaxSize",
            };

        var collectionRequest = new EnableMetricsCollectionRequest
        {
            AutoScalingGroupName = groupName,
            Metrics = listMetrics,
            Granularity = "1Minute",
        };

        var response = await
_amazonAutoScaling.EnableMetricsCollectionAsync(collectionRequest);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- Untuk detail API, lihat EnableMetricsCollectiondi Referensi AWS SDK untuk .NET API.

`SetDesiredCapacity`

Contoh kode berikut menunjukkan cara menggunakan`SetDesiredCapacity`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Set the desired capacity of an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <param name="desiredCapacity">The desired capacity for the Auto
    /// Scaling group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> SetDesiredCapacityAsync(
        string groupName,
        int desiredCapacity)
    {
        var capacityRequest = new SetDesiredCapacityRequest
        {
            AutoScalingGroupName = groupName,
            DesiredCapacity = desiredCapacity,
        };

        var response = await
_amazonAutoScaling.SetDesiredCapacityAsync(capacityRequest);
        Console.WriteLine($"You have set the DesiredCapacity to
{desiredCapacity}.");

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- Untuk detail API, lihat SetDesiredCapacitydi Referensi AWS SDK untuk .NET API.

`TerminateInstanceInAutoScalingGroup`

Contoh kode berikut menunjukkan cara menggunakan`TerminateInstanceInAutoScalingGroup`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Terminate all instances in the Auto Scaling group in preparation for
    /// deleting the group.
    /// </summary>
    /// <param name="instanceId">The instance Id of the instance to terminate.</
param>
    /// <returns>A Boolean value that indicates the success or failure of
    /// the operation.</returns>
    public async Task<bool> TerminateInstanceInAutoScalingGroupAsync(
        string instanceId)
    {
        var request = new TerminateInstanceInAutoScalingGroupRequest
        {
            InstanceId = instanceId,
            ShouldDecrementDesiredCapacity = false,
        };

        var response = await
 _amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"You have terminated the instance: {instanceId}");
            return true;
        }

        Console.WriteLine($"Could not terminate {instanceId}");
        return false;
    }
```

- Untuk detail API, lihat TerminateInstanceInAutoScalingGroupdi Referensi AWS SDK untuk .NET API.

`UpdateAutoScalingGroup`

Contoh kode berikut menunjukkan cara menggunakan`UpdateAutoScalingGroup`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Update the capacity of an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <param name="launchTemplateName">The name of the EC2 launch template.</
param>
    /// <param name="maxSize">The maximum number of instances that can be
    /// created for the Auto Scaling group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> UpdateAutoScalingGroupAsync(
        string groupName,
        string launchTemplateName,
        int maxSize)
    {
        var templateSpecification = new LaunchTemplateSpecification
        {
            LaunchTemplateName = launchTemplateName,
        };

        var groupRequest = new UpdateAutoScalingGroupRequest
        {
            MaxSize = maxSize,
            AutoScalingGroupName = groupName,
            LaunchTemplate = templateSpecification,
        };

        var response = await
 _amazonAutoScaling.UpdateAutoScalingGroupAsync(groupRequest);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
```

```
            Console.WriteLine($"You successfully updated the Auto Scaling group
{groupName}.");
            return true;
        }
        else
        {
            return false;
        }
    }
```

- Untuk detail API, lihat UpdateAutoScalingGroupdi Referensi AWS SDK untuk .NET API.

Contoh Amazon Bedrock menggunakan SDK untuk .NET (v4)

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK untuk .NET (v4) dengan Amazon Bedrock.

Tindakan merupakan 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

- Memulai
- Tindakan

Memulai

Halo Amazon Bedrock

Contoh kode berikut menunjukkan cara memulai menggunakan Amazon Bedrock.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
using Amazon;
using Amazon.Bedrock;
using Amazon.Bedrock.Model;

namespace BedrockActions;

/// <summary>
/// This example shows how to list foundation models.
/// </summary>
internal class HelloBedrock
{
    /// <summary>
    /// Main method to call the ListFoundationModelsAsync method.
    /// </summary>
    /// <param name="args"> The command line arguments. </param>
    static async Task Main(string[] args)
    {
        // Specify a region endpoint where Amazon Bedrock is available. For a list
 of supported region see https://docs.aws.amazon.com/bedrock/latest/userguide/what-
is-bedrock.html#bedrock-regions
        AmazonBedrockClient bedrockClient = new(RegionEndpoint.USWest2);

        await ListFoundationModelsAsync(bedrockClient);

    }


    /// <summary>
    /// List foundation models.
    /// </summary>
    /// <param name="bedrockClient"> The Amazon Bedrock client. </param>
    private static async Task ListFoundationModelsAsync(AmazonBedrockClient
 bedrockClient)
    {
```

```
        Console.WriteLine("List foundation models with no filter.");

        try
        {
            var response = await bedrockClient.ListFoundationModelsAsync(new
 ListFoundationModelsRequest()
            {
            });

            if (response?.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                foreach (var fm in response.ModelSummaries)
                {
                    WriteToConsole(fm);
                }
            }
            else
            {
                Console.WriteLine("Something wrong happened");
            }
        }
        catch (AmazonBedrockException e)
        {
            Console.WriteLine(e.Message);
        }
    }


    /// <summary>
    /// Write the foundation model summary to console.
    /// </summary>
    /// <param name="foundationModel"> The foundation model summary to write to
 console. </param>
    private static void WriteToConsole(FoundationModelSummary foundationModel)
    {
        Console.WriteLine($"{foundationModel.ModelId}, Customization:
 {string.Join(", ", foundationModel.CustomizationsSupported)}, Stream:
 {foundationModel.ResponseStreamingSupported}, Input: {string.Join(",
 ", foundationModel.InputModalities)}, Output: {string.Join(", ",
 foundationModel.OutputModalities)}");
    }
}
```

- Untuk detail API, lihat ListFoundationModelsdi Referensi AWS SDK untuk .NET API.

Tindakan

`ListFoundationModels`

Contoh kode berikut menunjukkan cara menggunakan`ListFoundationModels`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Buat daftar model foundation Bedrock yang tersedia.

```
   /// <summary>
   /// List foundation models.
   /// </summary>
   /// <param name="bedrockClient"> The Amazon Bedrock client. </param>
   private static async Task ListFoundationModelsAsync(AmazonBedrockClient
bedrockClient)
   {
       Console.WriteLine("List foundation models with no filter.");

       try
       {
           var response = await bedrockClient.ListFoundationModelsAsync(new
ListFoundationModelsRequest()
           {
           });

           if (response?.HttpStatusCode == System.Net.HttpStatusCode.OK)
           {
               foreach (var fm in response.ModelSummaries)
               {
                   WriteToConsole(fm);
               }
```

```
            }
            else
            {
                Console.WriteLine("Something wrong happened");
            }
        }
        catch (AmazonBedrockException e)
        {
            Console.WriteLine(e.Message);
        }
    }
```

- Untuk detail API, lihat ListFoundationModelsdi Referensi AWS SDK untuk .NET API.

Contoh Amazon Bedrock Runtime menggunakan SDK untuk .NET (v4)

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK untuk .NET (v4) dengan Amazon Bedrock Runtime.

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
- Perintah Cohere
- Meta Llama
- Mistral AI

Antropik Claude

Bercakap-cakap

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

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

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

```
// Use the Converse API to send a text message to Anthropic Claude.

using System;
using System.Collections.Generic;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Claude 3 Haiku.
var modelId = "anthropic.claude-3-haiku-20240307-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
```

```
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseAsync(request);

    // Extract and print the response text.
    string responseText = response?.Output?.Message?.Content?[0]?.Text ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Untuk detail API, lihat Converse di Referensi AWS SDK untuk .NET API.

ConverseStream

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

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Kirim pesan teks ke Anthropic Claude, menggunakan API Converse Bedrock dan proses aliran respons secara real-time.

```
// Use the Converse API to send a text message to Anthropic Claude
```

```
// and print the response stream.

using System;
using System.Collections.Generic;
using System.Linq;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Claude 3 Haiku.
var modelId = "anthropic.claude-3-haiku-20240307-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseStreamRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseStreamAsync(request);
```

```
    // Extract and print the streamed response text in real-time.
    foreach (var chunk in response.Stream.AsEnumerable())
    {
        if (chunk is ContentBlockDeltaEvent)
        {
            Console.Write((chunk as ContentBlockDeltaEvent).Delta.Text);
        }
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Untuk detail API, lihat ConverseStreamdi Referensi AWS SDK untuk .NET API.

InvokeModel

Contoh kode berikut menunjukkan cara mengirim pesan teks ke Anthropic Claude, menggunakan Invoke Model API.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Gunakan API Invoke Model untuk mengirim pesan teks.

```
// Use the native inference API to send a text message to Anthropic Claude.

using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;
using Amazon;
```

```
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Claude 3 Haiku.
var modelId = "anthropic.claude-3-haiku-20240307-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    anthropic_version = "bedrock-2023-05-31",
    max_tokens = 512,
    temperature = 0.5,
    messages = new[]
    {
        new { role = "user", content = userMessage }
    }
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelRequest()
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var response = await client.InvokeModelAsync(request);

    // Decode the response body.
    var modelResponse = await JsonNode.ParseAsync(response.Body);

    // Extract and print the response text.
    var responseText = modelResponse["content"]?[0]?["text"] ?? "";
    Console.WriteLine(responseText);
}
```

```
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Untuk detail API, lihat InvokeModeldi Referensi AWS SDK untuk .NET API.

Perintah Cohere

Bercakap-cakap

Contoh kode berikut menunjukkan cara mengirim pesan teks ke Cohere Command, menggunakan API Converse Bedrock.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Kirim pesan teks ke Cohere Command, menggunakan API Converse Bedrock.

```
// Use the Converse API to send a text message to Cohere Command.

using System;
using System.Collections.Generic;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Command R.
var modelId = "cohere.command-r-v1:0";
```

```
// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseAsync(request);

    // Extract and print the response text.
    string responseText = response?.Output?.Message?.Content?[0]?.Text ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Untuk detail API, lihat Converse di Referensi AWS SDK untuk .NET API.

ConverseStream

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

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Kirim pesan teks ke Cohere Command, menggunakan API Converse Bedrock dan proses aliran respons secara real-time.

```
// Use the Converse API to send a text message to Cohere Command
// and print the response stream.

using System;
using System.Collections.Generic;
using System.Linq;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Command R.
var modelId = "cohere.command-r-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseStreamRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
```

```
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var chunk in response.Stream.AsEnumerable())
    {
        if (chunk is ContentBlockDeltaEvent)
        {
            Console.Write((chunk as ContentBlockDeltaEvent).Delta.Text);
        }
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Untuk detail API, lihat ConverseStreamdi Referensi AWS SDK untuk .NET API.

Meta Llama

Bercakap-cakap

Contoh kode berikut menunjukkan cara mengirim pesan teks ke Meta Llama, menggunakan API Converse Bedrock.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Kirim pesan teks ke Meta Llama, menggunakan API Converse Bedrock.

```
// Use the Converse API to send a text message to Meta Llama.

using System;
using System.Collections.Generic;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Llama 3 8b Instruct.
var modelId = "meta.llama3-8b-instruct-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
```

```
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseAsync(request);

    // Extract and print the response text.
    string responseText = response?.Output?.Message?.Content?[0]?.Text ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Untuk detail API, lihat Converse di Referensi AWS SDK untuk .NET API.

ConverseStream

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

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Kirim pesan teks ke Meta Llama, menggunakan API Converse Bedrock dan proses aliran respons secara real-time.

```
// Use the Converse API to send a text message to Meta Llama
// and print the response stream.

using System;
using System.Collections.Generic;
using System.Linq;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Llama 3 8b Instruct.
var modelId = "meta.llama3-8b-instruct-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseStreamRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
```

```
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var chunk in response.Stream.AsEnumerable())
    {
        if (chunk is ContentBlockDeltaEvent)
        {
            Console.Write((chunk as ContentBlockDeltaEvent).Delta.Text);
        }
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Untuk detail API, lihat ConverseStreamdi Referensi AWS SDK untuk .NET API.

Mistral AI

Bercakap-cakap

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

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Kirim pesan teks ke Mistral, menggunakan API Converse Bedrock.

```
// Use the Converse API to send a text message to Mistral.

using System;
using System.Collections.Generic;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Mistral Large.
var modelId = "mistral.mistral-large-2402-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
```

```
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseAsync(request);

    // Extract and print the response text.
    string responseText = response?.Output?.Message?.Content?[0]?.Text ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Untuk detail API, lihat Converse di Referensi AWS SDK untuk .NET API.

ConverseStream

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

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Kirim pesan teks ke Mistral, menggunakan API Converse Bedrock dan proses aliran respons secara real-time.

```
// Use the Converse API to send a text message to Mistral
```

```
// and print the response stream.

using System;
using System.Collections.Generic;
using System.Linq;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Mistral Large.
var modelId = "mistral.mistral-large-2402-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseStreamRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseStreamAsync(request);
```

```
    // Extract and print the streamed response text in real-time.
    foreach (var chunk in response.Stream.AsEnumerable())
    {
        if (chunk is ContentBlockDeltaEvent)
        {
            Console.Write((chunk as ContentBlockDeltaEvent).Delta.Text);
        }
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Untuk detail API, lihat ConverseStreamdi Referensi AWS SDK untuk .NET API.

CloudFormation contoh menggunakan SDK untuk .NET (v4)

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK untuk .NET (v4) dengan CloudFormation.

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

Topik

- Memulai

Memulai

Halo CloudFormation

Contoh kode berikut menunjukkan bagaimana untuk mulai menggunakan CloudFormation.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
using Amazon.CloudFormation;
using Amazon.CloudFormation.Model;
using Amazon.Runtime;

namespace CloudFormationActions;

public static class HelloCloudFormation
{
    public static IAmazonCloudFormation _amazonCloudFormation = null!;

    static async Task Main(string[] args)
    {
        // Create the CloudFormation client
        _amazonCloudFormation = new AmazonCloudFormationClient();
        Console.WriteLine($"\nIn Region:
 {_amazonCloudFormation.Config.RegionEndpoint}");

        // List the resources for each stack
        await ListResources();
    }

    /// <summary>
    /// Method to list stack resources and other information.
    /// </summary>
    /// <returns>True if successful.</returns>
    public static async Task<bool> ListResources()
    {
        try
        {
            Console.WriteLine("Getting CloudFormation stack information...");

            // Get all stacks using the stack paginator.
            var paginatorForDescribeStacks =
                _amazonCloudFormation.Paginators.DescribeStacks(
```

```
                    new DescribeStacksRequest());
            if (paginatorForDescribeStacks.Stacks != null)
            {
                await foreach (Stack stack in paginatorForDescribeStacks.Stacks)
                {
                    // Basic information for each stack
                    Console.WriteLine(
                        "\n-----------------------------------------------");
                    Console.WriteLine($"\nStack: {stack.StackName}");
                    Console.WriteLine($"  Status: {stack.StackStatus.Value}");
                    Console.WriteLine($"  Created: {stack.CreationTime}");

                    // The tags of each stack (etc.)
                    if (stack.Tags != null && stack.Tags.Count > 0)
                    {
                        Console.WriteLine("  Tags:");
                        foreach (Tag tag in stack.Tags)
                            Console.WriteLine($"    {tag.Key}, {tag.Value}");
                    }

                    // The resources of each stack
                    DescribeStackResourcesResponse responseDescribeResources =
                        await _amazonCloudFormation.DescribeStackResourcesAsync(
                            new DescribeStackResourcesRequest
                            {
                                StackName = stack.StackName
                            });
                    if (responseDescribeResources.StackResources != null &&
responseDescribeResources.StackResources.Count > 0)
                    {
                        Console.WriteLine("  Resources:");
                        foreach (StackResource resource in responseDescribeResources
                                    .StackResources)
                            Console.WriteLine(
                                $"    {resource.LogicalResourceId}:
{resource.ResourceStatus}");
                    }
                }
            }

            Console.WriteLine("\n-----------------------------------------------");
            return true;
        }
        catch (AmazonCloudFormationException ex)
```

```
        {
            Console.WriteLine("Unable to get stack information:\n" + ex.Message);
            return false;
        }
        catch (AmazonServiceException ex)
        {
            if (ex.Message.Contains("Unable to get IAM security credentials"))
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine("If you are usnig SSO, be sure to install" +
                                  " the AWSSDK.SSO and AWSSDK.SSOOIDC packages.");
            }
            else
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine(ex.StackTrace);
            }

            return false;
        }
        catch (ArgumentNullException ex)
        {
            if (ex.Message.Contains("Options property cannot be empty: ClientName"))
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine("If you are using SSO, have you logged in?");
            }
            else
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine(ex.StackTrace);
            }

            return false;
        }
    }
```

- Untuk detail API, lihat DescribeStackResourcesdi Referensi AWS SDK untuk .NET API.

CloudWatch contoh menggunakan SDK untuk .NET (v4)

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK untuk .NET (v4) dengan CloudWatch.

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

Tindakan merupakan 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

- Memulai
- Hal-hal mendasar
- Tindakan

Memulai

Halo CloudWatch

Contoh kode berikut menunjukkan bagaimana untuk mulai menggunakan CloudWatch.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
using Amazon.CloudWatch;
using Amazon.CloudWatch.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
```

```
namespace CloudWatchActions;

public static class HelloCloudWatch
{
    static async Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
 the Amazon CloudWatch service.
        // Use your AWS profile name, or leave it blank to use the default profile.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonCloudWatch>()
            ).Build();

        // Now the client is available for injection.
        var cloudWatchClient =
 host.Services.GetRequiredService<IAmazonCloudWatch>();

        // You can use await and any of the async methods to get a response.
        var metricNamespace = "AWS/Billing";
        var response = await cloudWatchClient.ListMetricsAsync(new
 ListMetricsRequest
        {
            Namespace = metricNamespace
        });
        Console.WriteLine($"Hello Amazon CloudWatch! Following are some metrics
 available in the {metricNamespace} namespace:");
        Console.WriteLine();
        if (response.Metrics != null)
        {
            foreach (var metric in response.Metrics.Take(5))
            {
                Console.WriteLine($"\tMetric: {metric.MetricName}");
                Console.WriteLine($"\tNamespace: {metric.Namespace}");
                Console.WriteLine(
                    $"\tDimensions: {string.Join(", ", metric.Dimensions.Select(m =>
 $"{m.Name}:{m.Value}"))}");
                Console.WriteLine();
            }
        }
    }
}
```

- Untuk detail API, lihat ListMetricsdi Referensi AWS SDK untuk .NET API.

Hal-hal mendasar

Pelajari dasar-dasarnya

Contoh kode berikut ini menunjukkan cara untuk melakukan:

- Buat daftar CloudWatch ruang nama dan metrik.
- Ambil statistik untuk metrik dan estimasi penagihan.
- Membuat dan memperbarui sebuah dasbor.
- Membuat dan menambahkan data ke metrik.
- Membuat dan memicu alarm, lalu lihat riwayat alarm.
- Menambahkan detektor anomali.
- Ambil gambar metrik, lalu bersihkan sumber daya.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkap dan pelajari cara menyiapkan dan menjalankan di Repositori Contoh Kode AWS.

Jalankan skenario interaktif di penggugah/prompt perintah.

```
public class CloudWatchScenario
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.

    To enable billing metrics and statistics for this example, make sure billing
 alerts are enabled for your account:
    https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
monitor_estimated_charges_with_cloudwatch.html#turning_on_billing_metrics

    This .NET example performs the following tasks:
```

```
        1. List and select a CloudWatch namespace.
        2. List and select a CloudWatch metric.
        3. Get statistics for a CloudWatch metric.
        4. Get estimated billing statistics for the last week.
        5. Create a new CloudWatch dashboard with two metrics.
        6. List current CloudWatch dashboards.
        7. Create a CloudWatch custom metric and add metric data.
        8. Add the custom metric to the dashboard.
        9. Create a CloudWatch alarm for the custom metric.
       10. Describe current CloudWatch alarms.
       11. Get recent data for the custom metric.
       12. Add data to the custom metric to trigger the alarm.
       13. Wait for an alarm state.
       14. Get history for the CloudWatch alarm.
       15. Add an anomaly detector.
       16. Describe current anomaly detectors.
       17. Get and display a metric image.
       18. Clean up resources.
   */

   private static ILogger logger = null!;
   private static CloudWatchWrapper _cloudWatchWrapper = null!;
   private static IConfiguration _configuration = null!;
   private static readonly List<string> _statTypes = new List<string>
{ "SampleCount", "Average", "Sum", "Minimum", "Maximum" };
   private static SingleMetricAnomalyDetector? anomalyDetector = null!;

   static async Task Main(string[] args)
   {
       // Set up dependency injection for the Amazon service.
       using var host = Host.CreateDefaultBuilder(args)
           .ConfigureLogging(logging =>
               logging.AddFilter("System", LogLevel.Debug)
                   .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                   .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
           .ConfigureServices((_, services) =>
           services.AddAWSService<IAmazonCloudWatch>()
           .AddTransient<CloudWatchWrapper>()
       )
       .Build();

       _configuration = new ConfigurationBuilder()
           .SetBasePath(Directory.GetCurrentDirectory())
```

```
            .AddJsonFile("settings.json") // Load settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally, load local settings.
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<CloudWatchScenario>();

        _cloudWatchWrapper = host.Services.GetRequiredService<CloudWatchWrapper>();

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the Amazon CloudWatch example scenario.");
        Console.WriteLine(new string('-', 80));

        try
        {
            var selectedNamespace = await SelectNamespace();
            var selectedMetric = await SelectMetric(selectedNamespace);
            await GetAndDisplayMetricStatistics(selectedNamespace, selectedMetric);
            await GetAndDisplayEstimatedBilling();
            await CreateDashboardWithMetrics();
            await ListDashboards();
            await CreateNewCustomMetric();
            await AddMetricToDashboard();
            await CreateMetricAlarm();
            await DescribeAlarms();
            await GetCustomMetricData();
            await AddMetricDataForAlarm();
            await CheckForMetricAlarm();
            await GetAlarmHistory();
            anomalyDetector = await AddAnomalyDetector();
            await DescribeAnomalyDetectors();
            await GetAndOpenMetricImage();
            await CleanupResources();
        }
        catch (Exception ex)
        {
            logger.LogError(ex, "There was a problem executing the scenario.");
            await CleanupResources();
        }

    }

    /// <summary>
```

```
    /// Select a namespace.
    /// </summary>
    /// <returns>The selected namespace.</returns>
    private static async Task<string> SelectNamespace()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"1. Select a CloudWatch Namespace from a list of
Namespaces.");
        var metrics = await _cloudWatchWrapper.ListMetrics();
        // Get a distinct list of namespaces.
        var namespaces = metrics.Select(m => m.Namespace).Distinct().ToList();
        for (int i = 0; i < namespaces.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {namespaces[i]}");
        }

        var namespaceChoiceNumber = 0;
        while (namespaceChoiceNumber < 1 || namespaceChoiceNumber >
namespaces.Count)
        {
            Console.WriteLine(
                "Select a namespace by entering a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out namespaceChoiceNumber);
        }

        var selectedNamespace = namespaces[namespaceChoiceNumber - 1];

        Console.WriteLine(new string('-', 80));

        return selectedNamespace;
    }

    /// <summary>
    /// Select a metric from a namespace.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <returns>The metric name.</returns>
    private static async Task<Metric> SelectMetric(string metricNamespace)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"2. Select a CloudWatch metric from a namespace.");
```

```
        var namespaceMetrics = await
_cloudWatchWrapper.ListMetrics(metricNamespace);

        for (int i = 0; i < namespaceMetrics.Count && i < 15; i++)
        {
            var dimensionsWithValues = namespaceMetrics[i].Dimensions
                .Where(d => !string.Equals("None", d.Value));
            Console.WriteLine($"\t{i + 1}. {namespaceMetrics[i].MetricName} " +
                              $"{string.Join(", :", dimensionsWithValues.Select(d =>
d.Value))}");
        }

        var metricChoiceNumber = 0;
        while (metricChoiceNumber < 1 || metricChoiceNumber >
namespaceMetrics.Count)
        {
            Console.WriteLine(
                "Select a metric by entering a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out metricChoiceNumber);
        }

        var selectedMetric = namespaceMetrics[metricChoiceNumber - 1];

        Console.WriteLine(new string('-', 80));

        return selectedMetric;
    }

    /// <summary>
    /// Get and display metric statistics for a specific metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task GetAndDisplayMetricStatistics(string metricNamespace,
Metric metric)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"3. Get CloudWatch metric statistics for the last day.");

        for (int i = 0; i < _statTypes.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {_statTypes[i]}");
```

```
        }

        var statisticChoiceNumber = 0;
        while (statisticChoiceNumber < 1 || statisticChoiceNumber >
_statTypes.Count)
        {
            Console.WriteLine(
                "Select a metric statistic by entering a number from the preceding
list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out statisticChoiceNumber);
        }

        var selectedStatistic = _statTypes[statisticChoiceNumber - 1];
        var statisticsList = new List<string> { selectedStatistic };

        var metricStatistics = await
_cloudWatchWrapper.GetMetricStatistics(metricNamespace, metric.MetricName,
statisticsList, metric.Dimensions, 1, 60);

        if (!metricStatistics.Any())
        {
            Console.WriteLine($"No {selectedStatistic} statistics found for {metric}
in namespace {metricNamespace}.");
        }

        metricStatistics = metricStatistics.OrderBy(s => s.Timestamp).ToList();
        for (int i = 0; i < metricStatistics.Count && i < 10; i++)
        {
            var metricStat = metricStatistics[i];
            var statValue =
metricStat.GetType().GetProperty(selectedStatistic)!.GetValue(metricStat, null);
            Console.WriteLine($"\t{i + 1}. Timestamp
{metricStatistics[i].Timestamp:G} {selectedStatistic}: {statValue}");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get and display estimated billing statistics.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
```

```
    /// <returns>Async task.</returns>
    private static async Task GetAndDisplayEstimatedBilling()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"4. Get CloudWatch estimated billing for the last
week.");

        var billingStatistics = await SetupBillingStatistics();

        for (int i = 0; i < billingStatistics.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. Timestamp
{billingStatistics[i].Timestamp:G} : {billingStatistics[i].Maximum}");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get billing statistics using a call to a wrapper class.
    /// </summary>
    /// <returns>A collection of billing statistics.</returns>
    private static async Task<List<Datapoint>> SetupBillingStatistics()
    {
        // Make a request for EstimatedCharges with a period of one day for the past
seven days.
        var billingStatistics = await _cloudWatchWrapper.GetMetricStatistics(
            "AWS/Billing",
            "EstimatedCharges",
            new List<string>() { "Maximum" },
            new List<Dimension>() { new Dimension { Name = "Currency", Value =
"USD" } },
            7,
            86400);

        billingStatistics = billingStatistics.OrderBy(n => n.Timestamp).ToList();

        return billingStatistics;
    }

    /// <summary>
    /// Create a dashboard with metrics.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
```

```
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task CreateDashboardWithMetrics()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"5. Create a new CloudWatch dashboard with metrics.");
        var dashboardName = _configuration["dashboardName"];
        var newDashboard = new DashboardModel();
        _configuration.GetSection("dashboardExampleBody").Bind(newDashboard);
        var newDashboardString = JsonSerializer.Serialize(
            newDashboard,
            new JsonSerializerOptions
            {
                DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull
            });
        var validationMessages =
            await _cloudWatchWrapper.PutDashboard(dashboardName,
newDashboardString);

        Console.WriteLine(validationMessages.Any() ? $"\tValidation messages:" :
null);
        for (int i = 0; i < validationMessages.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {validationMessages[i].Message}");
        }
        Console.WriteLine($"\tDashboard {dashboardName} was created.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List dashboards.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task ListDashboards()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"6. List the CloudWatch dashboards in the current
account.");

        var dashboards = await _cloudWatchWrapper.ListDashboards();

        for (int i = 0; i < dashboards.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {dashboards[i].DashboardName}");
```

```
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Create and add data for a new custom metric.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CreateNewCustomMetric()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"7. Create and add data for a new custom metric.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var customData = await PutRandomMetricData(customMetricName,
customMetricNamespace);

        var valuesString = string.Join(',', customData.Select(d => d.Value));
        Console.WriteLine($"\tAdded metric values for for metric {customMetricName}:
\n\t{valuesString}");

        Console.WriteLine(new string('-', 80));
    }


    /// <summary>
    /// Add some metric data using a call to a wrapper class.
    /// </summary>
    /// <param name="customMetricName">The metric name.</param>
    /// <param name="customMetricNamespace">The metric namespace.</param>
    /// <returns></returns>
    private static async Task<List<MetricDatum>> PutRandomMetricData(string
customMetricName,
        string customMetricNamespace)
    {
        List<MetricDatum> customData = new List<MetricDatum>();
        Random rnd = new Random();

        // Add 10 random values up to 100, starting with a timestamp 15 minutes in
the past.
        var utcNowMinus15 = DateTime.UtcNow.AddMinutes(-15);
```

```
        for (int i = 0; i < 10; i++)
        {
            var metricValue = rnd.Next(0, 100);
            customData.Add(
                new MetricDatum
                {
                    MetricName = customMetricName,
                    Value = metricValue,
                    TimestampUtc = utcNowMinus15.AddMinutes(i)
                }
            );
        }

        await _cloudWatchWrapper.PutMetricData(customMetricNamespace, customData);
        return customData;
    }

    /// <summary>
    /// Add the custom metric to the dashboard.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task AddMetricToDashboard()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"8. Add the new custom metric to the dashboard.");

        var dashboardName = _configuration["dashboardName"];

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var validationMessages = await SetupDashboard(customMetricNamespace,
customMetricName, dashboardName);

        Console.WriteLine(validationMessages.Any() ? $"\tValidation messages:" :
null);
        for (int i = 0; i < validationMessages.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {validationMessages[i].Message}");
        }
        Console.WriteLine($"\tDashboard {dashboardName} updated with metric
{customMetricName}.");
        Console.WriteLine(new string('-', 80));
    }
```

```
/// <summary>
/// Set up a dashboard using a call to the wrapper class.
/// </summary>
/// <param name="customMetricNamespace">The metric namespace.</param>
/// <param name="customMetricName">The metric name.</param>
/// <param name="dashboardName">The name of the dashboard.</param>
/// <returns>A list of validation messages.</returns>
private static async Task<List<DashboardValidationMessage>> SetupDashboard(
    string customMetricNamespace, string customMetricName, string dashboardName)
{
    // Get the dashboard model from configuration.
    var newDashboard = new DashboardModel();
    _configuration.GetSection("dashboardExampleBody").Bind(newDashboard);

    // Add a new metric to the dashboard.
    newDashboard.Widgets.Add(new Widget
    {
        Height = 8,
        Width = 8,
        Y = 8,
        X = 0,
        Type = "metric",
        Properties = new Properties
        {
            Metrics = new List<List<object>>
                { new() { customMetricNamespace, customMetricName } },
            View = "timeSeries",
            Region = "us-east-1",
            Stat = "Sum",
            Period = 86400,
            YAxis = new YAxis { Left = new Left { Min = 0, Max = 100 } },
            Title = "Custom Metric Widget",
            LiveData = true,
            Sparkline = true,
            Trend = true,
            Stacked = false,
            SetPeriodToTimeRange = false
        }
    });

    var newDashboardString = JsonSerializer.Serialize(newDashboard,
        new JsonSerializerOptions
```

```
            { DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull });
        var validationMessages =
            await _cloudWatchWrapper.PutDashboard(dashboardName,
newDashboardString);

        return validationMessages;
    }

    /// <summary>
    /// Create a CloudWatch alarm for the new metric.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CreateMetricAlarm()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"9. Create a CloudWatch alarm for the new metric.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var alarmName = _configuration["exampleAlarmName"];
        var accountId = _configuration["accountId"];
        var region = _configuration["region"];
        var emailTopic = _configuration["emailTopic"];
        var alarmActions = new List<string>();

        if (GetYesNoResponse(
                $"\tAdd an email action for topic {emailTopic} to alarm {alarmName}?
(y/n)"))
        {
            _cloudWatchWrapper.AddEmailAlarmAction(accountId, region, emailTopic,
alarmActions);
        }

        await _cloudWatchWrapper.PutMetricEmailAlarm(
            "Example metric alarm",
            alarmName,
            ComparisonOperator.GreaterThanOrEqualToThreshold,
            customMetricName,
            customMetricNamespace,
            100,
            alarmActions);
```

```
        Console.WriteLine($"\tAlarm {alarmName} added for metric
{customMetricName}.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Describe Alarms.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task DescribeAlarms()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"10. Describe CloudWatch alarms in the current
account.");

        var alarms = await _cloudWatchWrapper.DescribeAlarms();
        alarms = alarms.OrderByDescending(a => a.StateUpdatedTimestamp).ToList();

        for (int i = 0; i < alarms.Count && i < 10; i++)
        {
            var alarm = alarms[i];
            Console.WriteLine($"\t{i + 1}. {alarm.AlarmName}");
            Console.WriteLine($"\tState: {alarm.StateValue} for {alarm.MetricName}
{alarm.ComparisonOperator} {alarm.Threshold}");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get the recent data for the metric.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetCustomMetricData()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"11. Get current data for new custom metric.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];
        var accountId = _configuration["accountId"];

        var query = new List<MetricDataQuery>
        {
```

```
            new MetricDataQuery
            {
                AccountId = accountId,
                Id = "m1",
                Label = "Custom Metric Data",
                MetricStat = new MetricStat
                {
                    Metric = new Metric
                    {
                        MetricName = customMetricName,
                        Namespace = customMetricNamespace,
                    },
                    Period = 1,
                    Stat = "Maximum"
                }
            }
        };

        var metricData = await _cloudWatchWrapper.GetMetricData(
            20,
            true,
            DateTime.UtcNow.AddMinutes(1),
            20,
            query);

        for (int i = 0; i < metricData.Count; i++)
        {
            if (metricData[i].Values != null)
            {
                for (int j = 0; j < metricData[i].Values.Count; j++)
                {
                    Console.WriteLine(
                        $"\tTimestamp {metricData[i].Timestamps[j]:G} Value:
{metricData[i].Values[j]}");
                }
            }
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Add metric data to trigger an alarm.
    /// </summary>
```

```
    /// <returns>Async task.</returns>
    private static async Task AddMetricDataForAlarm()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"12. Add metric data to the custom metric to trigger an
alarm.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];
        var nowUtc = DateTime.UtcNow;
        List<MetricDatum> customData = new List<MetricDatum>
        {
            new MetricDatum
            {
                MetricName = customMetricName,
                Value = 101,
                TimestampUtc = nowUtc.AddMinutes(-2)
            },
            new MetricDatum
            {
                MetricName = customMetricName,
                Value = 101,
                TimestampUtc = nowUtc.AddMinutes(-1)
            },
            new MetricDatum
            {
                MetricName = customMetricName,
                Value = 101,
                TimestampUtc = nowUtc
            }
        };
        var valuesString = string.Join(',', customData.Select(d => d.Value));
        Console.WriteLine($"\tAdded metric values for for metric {customMetricName}:
\n\t{valuesString}");
        await _cloudWatchWrapper.PutMetricData(customMetricNamespace, customData);

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Check for a metric alarm using the DescribeAlarmsForMetric action.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CheckForMetricAlarm()
```

```
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"13. Checking for an alarm state.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];
        var hasAlarm = false;
        var retries = 10;
        while (!hasAlarm && retries > 0)
        {
            var alarms = await
_cloudWatchWrapper.DescribeAlarmsForMetric(customMetricNamespace,
customMetricName);
            hasAlarm = alarms.Any(a => a.StateValue == StateValue.ALARM);
            retries--;
            Thread.Sleep(20000);
        }

        Console.WriteLine(hasAlarm
            ? $"\tAlarm state found for {customMetricName}."
            : $"\tNo Alarm state found for {customMetricName} after 10 retries.");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get history for an alarm.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetAlarmHistory()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"14. Get alarm history.");

        var exampleAlarmName = _configuration["exampleAlarmName"];

        var alarmHistory = await
_cloudWatchWrapper.DescribeAlarmHistory(exampleAlarmName, 2);

        for (int i = 0; i < alarmHistory.Count; i++)
        {
            var history = alarmHistory[i];
            Console.WriteLine($"\t{i + 1}. {history.HistorySummary}, time
{history.Timestamp:g}");
```

```
        }
        if (!alarmHistory.Any())
        {
            Console.WriteLine($"\tNo alarm history data found for
{exampleAlarmName}.");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Add an anomaly detector.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task<SingleMetricAnomalyDetector> AddAnomalyDetector()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"15. Add an anomaly detector.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var detector = new SingleMetricAnomalyDetector
        {
            MetricName = customMetricName,
            Namespace = customMetricNamespace,
            Stat = "Maximum"
        };
        await _cloudWatchWrapper.PutAnomalyDetector(detector);
        Console.WriteLine($"\tAdded anomaly detector for metric
{customMetricName}.");

        Console.WriteLine(new string('-', 80));
        return detector;
    }

    /// <summary>
    /// Describe anomaly detectors.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task DescribeAnomalyDetectors()
    {
        Console.WriteLine(new string('-', 80));
```

```
        Console.WriteLine($"16. Describe anomaly detectors in the current
account.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var detectors = await
_cloudWatchWrapper.DescribeAnomalyDetectors(customMetricNamespace,
customMetricName);

        for (int i = 0; i < detectors.Count; i++)
        {
            var detector = detectors[i];
            Console.WriteLine($"\t{i + 1}.
{detector.SingleMetricAnomalyDetector.MetricName}, state {detector.StateValue}");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Fetch and open a metrics image for a CloudWatch metric and namespace.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetAndOpenMetricImage()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("17. Get a metric image from CloudWatch.");

        Console.WriteLine($"\tGetting Image data for custom metric.");
        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var memoryStream = await
_cloudWatchWrapper.GetTimeSeriesMetricImage(customMetricNamespace,
customMetricName, "Maximum", 10);
        var file = _cloudWatchWrapper.SaveMetricImage(memoryStream, "MetricImages");

        ProcessStartInfo info = new ProcessStartInfo();

        Console.WriteLine($"\tFile saved as {Path.GetFileName(file)}.");
        Console.WriteLine($"\tPress enter to open the image.");
        Console.ReadLine();
        info.FileName = Path.Combine("ms-photos://", file);
```

```
        info.UseShellExecute = true;
        info.CreateNoWindow = true;
        info.Verb = string.Empty;

        Process.Start(info);

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Clean up created resources.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task CleanupResources()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"18. Clean up resources.");

        var dashboardName = _configuration["dashboardName"];
        if (GetYesNoResponse($"\tDelete dashboard {dashboardName}? (y/n)"))
        {
            Console.WriteLine($"\tDeleting dashboard.");
            var dashboardList = new List<string> { dashboardName };
            await _cloudWatchWrapper.DeleteDashboards(dashboardList);
        }

        var alarmName = _configuration["exampleAlarmName"];
        if (GetYesNoResponse($"\tDelete alarm {alarmName}? (y/n)"))
        {
            Console.WriteLine($"\tCleaning up alarms.");
            var alarms = new List<string> { alarmName };
            await _cloudWatchWrapper.DeleteAlarms(alarms);
        }

        if (GetYesNoResponse($"\tDelete anomaly detector? (y/n)") &&
anomalyDetector != null)
        {
            Console.WriteLine($"\tCleaning up anomaly detector.");

            await _cloudWatchWrapper.DeleteAnomalyDetector(
                anomalyDetector);
        }
```

```
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null &&
                       ynResponse.Equals("y",
                           StringComparison.InvariantCultureIgnoreCase);
        return response;
    }
}
```

Metode pembungkus yang digunakan oleh skenario untuk CloudWatch tindakan.

```
/// <summary>
/// Wrapper class for Amazon CloudWatch methods.
/// </summary>
public class CloudWatchWrapper
{
    private readonly IAmazonCloudWatch _amazonCloudWatch;
    private readonly ILogger<CloudWatchWrapper> _logger;

    /// <summary>
    /// Constructor for the CloudWatch wrapper.
    /// </summary>
    /// <param name="amazonCloudWatch">The injected CloudWatch client.</param>
    /// <param name="logger">The injected logger for the wrapper.</param>
    public CloudWatchWrapper(IAmazonCloudWatch amazonCloudWatch,
 ILogger<CloudWatchWrapper> logger)

    {
        _logger = logger;
        _amazonCloudWatch = amazonCloudWatch;
    }
```

```
   /// <summary>
   /// List metrics available, optionally within a namespace.
   /// </summary>
   /// <param name="metricNamespace">Optional CloudWatch namespace to use when
listing metrics.</param>
   /// <param name="filter">Optional dimension filter.</param>
   /// <param name="metricName">Optional metric name filter.</param>
   /// <returns>The list of metrics.</returns>
   public async Task<List<Metric>> ListMetrics(string? metricNamespace = null,
DimensionFilter? filter = null, string? metricName = null)
   {
       var results = new List<Metric>();
       var paginateMetrics = _amazonCloudWatch.Paginators.ListMetrics(
           new ListMetricsRequest
           {
               Namespace = metricNamespace,
               Dimensions = filter != null ? new List<DimensionFilter> { filter } :
null,
               MetricName = metricName
           });
       // Get the entire list using the paginator.
       await foreach (var metric in paginateMetrics.Metrics)
       {
           results.Add(metric);
       }

       return results;
   }

   /// <summary>
   /// Wrapper to get statistics for a specific CloudWatch metric.
   /// </summary>
   /// <param name="metricNamespace">The namespace of the metric.</param>
   /// <param name="metricName">The name of the metric.</param>
   /// <param name="statistics">The list of statistics to include.</param>
   /// <param name="dimensions">The list of dimensions to include.</param>
   /// <param name="days">The number of days in the past to include.</param>
   /// <param name="period">The period for the data.</param>
   /// <returns>A list of DataPoint objects for the statistics.</returns>
   public async Task<List<Datapoint>> GetMetricStatistics(string metricNamespace,
       string metricName, List<string> statistics, List<Dimension> dimensions, int
days, int period)
   {
```

```
        var metricStatistics = await _amazonCloudWatch.GetMetricStatisticsAsync(
            new GetMetricStatisticsRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName,
                Dimensions = dimensions,
                Statistics = statistics,
                StartTimeUtc = DateTime.UtcNow.AddDays(-days),
                EndTimeUtc = DateTime.UtcNow,
                Period = period
            });

        return metricStatistics.Datapoints ?? new List<Datapoint>();
    }

    /// <summary>
    /// Wrapper to create or add to a dashboard with metrics.
    /// </summary>
    /// <param name="dashboardName">The name for the dashboard.</param>
    /// <param name="dashboardBody">The metric data in JSON for the dashboard.</
param>
    /// <returns>A list of validation messages for the dashboard.</returns>
    public async Task<List<DashboardValidationMessage>> PutDashboard(string
 dashboardName,
        string dashboardBody)
    {
        // Updating a dashboard replaces all contents.
        // Best practice is to include a text widget indicating this dashboard was
 created programmatically.
        var dashboardResponse = await _amazonCloudWatch.PutDashboardAsync(
            new PutDashboardRequest()
            {
                DashboardName = dashboardName,
                DashboardBody = dashboardBody
            });

        return dashboardResponse.DashboardValidationMessages ?? new
 List<DashboardValidationMessage>();
    }


    /// <summary>
    /// Get information on a dashboard.
    /// </summary>
```

```
    /// <param name="dashboardName">The name of the dashboard.</param>
    /// <returns>A JSON object with dashboard information.</returns>
    public async Task<string> GetDashboard(string dashboardName)
    {
        var dashboardResponse = await _amazonCloudWatch.GetDashboardAsync(
            new GetDashboardRequest()
            {
                DashboardName = dashboardName
            });

        return dashboardResponse.DashboardBody;
    }


    /// <summary>
    /// Get a list of dashboards.
    /// </summary>
    /// <returns>A list of DashboardEntry objects.</returns>
    public async Task<List<DashboardEntry>> ListDashboards()
    {
        var results = new List<DashboardEntry>();
        var paginateDashboards = _amazonCloudWatch.Paginators.ListDashboards(
            new ListDashboardsRequest());
        // Get the entire list using the paginator.
        await foreach (var data in paginateDashboards.DashboardEntries)
        {
            results.Add(data);
        }

        return results;
    }

    /// <summary>
    /// Wrapper to add metric data to a CloudWatch metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricData">A data object for the metric data.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutMetricData(string metricNamespace,
        List<MetricDatum> metricData)
    {
        var putDataResponse = await _amazonCloudWatch.PutMetricDataAsync(
            new PutMetricDataRequest()
            {
```

```
                MetricData = metricData,
                Namespace = metricNamespace,
            });

        return putDataResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Get an image for a metric graphed over time.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metric">The name of the metric.</param>
    /// <param name="stat">The name of the stat to chart.</param>
    /// <param name="period">The period to use for the chart.</param>
    /// <returns>A memory stream for the chart image.</returns>
    public async Task<MemoryStream> GetTimeSeriesMetricImage(string metricNamespace,
string metric, string stat, int period)
    {
        var metricImageWidget = new
        {
            title = "Example Metric Graph",
            view = "timeSeries",
            stacked = false,
            period = period,
            width = 1400,
            height = 600,
            metrics = new List<List<object>>
                { new() { metricNamespace, metric, new { stat } } }
        };

        var metricImageWidgetString = JsonSerializer.Serialize(metricImageWidget);
        var imageResponse = await _amazonCloudWatch.GetMetricWidgetImageAsync(
            new GetMetricWidgetImageRequest()
            {
                MetricWidget = metricImageWidgetString
            });

        return imageResponse.MetricWidgetImage;
    }

    /// <summary>
    /// Save a metric image to a file.
    /// </summary>
    /// <param name="memoryStream">The MemoryStream for the metric image.</param>
```

```
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The path to the file.</returns>
    public string SaveMetricImage(MemoryStream memoryStream, string metricName)
    {
        var metricFileName = $"{metricName}_{DateTime.Now.Ticks}.png";
        using var sr = new StreamReader(memoryStream);
        // Writes the memory stream to a file.
        File.WriteAllBytes(metricFileName, memoryStream.ToArray());
        var filePath = Path.Join(AppDomain.CurrentDomain.BaseDirectory,
            metricFileName);
        return filePath;
    }

    /// <summary>
    /// Get data for CloudWatch metrics.
    /// </summary>
    /// <param name="minutesOfData">The number of minutes of data to include.</
param>
    /// <param name="useDescendingTime">True to return the data descending by
 time.</param>
    /// <param name="endDateUtc">The end date for the data, in UTC.</param>
    /// <param name="maxDataPoints">The maximum data points to include.</param>
    /// <param name="dataQueries">Optional data queries to include.</param>
    /// <returns>A list of the requested metric data.</returns>
    public async Task<List<MetricDataResult>> GetMetricData(int minutesOfData, bool
 useDescendingTime, DateTime? endDateUtc = null,
        int maxDataPoints = 0, List<MetricDataQuery>? dataQueries = null)
    {
        var metricData = new List<MetricDataResult>();
        // If no end time is provided, use the current time for the end time.
        endDateUtc ??= DateTime.UtcNow;
        var timeZoneOffset =
 TimeZoneInfo.Local.GetUtcOffset(endDateUtc.Value.ToLocalTime());
        var startTimeUtc = endDateUtc.Value.AddMinutes(-minutesOfData);
        // The timezone string should be in the format +0000, so use the timezone
 offset to format it correctly.
        var timeZoneString = $"{timeZoneOffset.Hours:D2}
{timeZoneOffset.Minutes:D2}";
        // Add the plus sign for positive offsets.
        timeZoneString = timeZoneString.StartsWith('-') ? timeZoneString : "+" +
 timeZoneString;
        var paginatedMetricData = _amazonCloudWatch.Paginators.GetMetricData(
            new GetMetricDataRequest()
            {
```

```
                StartTimeUtc = startTimeUtc,
                EndTimeUtc = endDateUtc.Value,
                LabelOptions = new LabelOptions { Timezone = timeZoneString },
                ScanBy = useDescendingTime ? ScanBy.TimestampDescending :
ScanBy.TimestampAscending,
                MaxDatapoints = maxDataPoints,
                MetricDataQueries = dataQueries,
            });

        if (paginatedMetricData.MetricDataResults != null)
        {
            await foreach (var data in paginatedMetricData.MetricDataResults)
            {
                metricData.Add(data);
            }
        }

        return metricData;
    }

    /// <summary>
    /// Add a metric alarm to send an email when the metric passes a threshold.
    /// </summary>
    /// <param name="alarmDescription">A description of the alarm.</param>
    /// <param name="alarmName">The name for the alarm.</param>
    /// <param name="comparison">The type of comparison to use.</param>
    /// <param name="metricName">The name of the metric for the alarm.</param>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="threshold">The threshold value for the alarm.</param>
    /// <param name="alarmActions">Optional actions to execute when in an alarm
state.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutMetricEmailAlarm(string alarmDescription, string
alarmName, ComparisonOperator comparison,
        string metricName, string metricNamespace, double threshold, List<string>
alarmActions = null!)
    {
        try
        {
            var putEmailAlarmResponse = await _amazonCloudWatch.PutMetricAlarmAsync(
                new PutMetricAlarmRequest()
                {
                    AlarmActions = alarmActions,
                    AlarmDescription = alarmDescription,
```

```
                    AlarmName = alarmName,
                    ComparisonOperator = comparison,
                    Threshold = threshold,
                    Namespace = metricNamespace,
                    MetricName = metricName,
                    EvaluationPeriods = 1,
                    Period = 10,
                    Statistic = new Statistic("Maximum"),
                    DatapointsToAlarm = 1,
                    TreatMissingData = "ignore"
                });
            return putEmailAlarmResponse.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (LimitExceededException lex)
        {
            _logger.LogError(lex, $"Unable to add alarm {alarmName}. Alarm quota has
already been reached.");
        }

        return false;
    }

    /// <summary>
    /// Add specific email actions to a list of action strings for a CloudWatch
alarm.
    /// </summary>
    /// <param name="accountId">The AccountId for the alarm.</param>
    /// <param name="region">The region for the alarm.</param>
    /// <param name="emailTopicName">An Amazon Simple Notification Service (SNS)
topic for the alarm email.</param>
    /// <param name="alarmActions">Optional list of existing alarm actions to append
to.</param>
    /// <returns>A list of string actions for an alarm.</returns>
    public List<string> AddEmailAlarmAction(string accountId, string region,
        string emailTopicName, List<string>? alarmActions = null)
    {
        alarmActions ??= new List<string>();
        var snsAlarmAction = $"arn:aws:sns:{region}:{accountId}:{emailTopicName}";
        alarmActions.Add(snsAlarmAction);
        return alarmActions;
    }

    /// <summary>
    /// Describe the current alarms, optionally filtered by state.
```

```
    /// </summary>
    /// <param name="stateValue">Optional filter for alarm state.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarms(StateValue? stateValue =
null)
    {
        List<MetricAlarm> alarms = new List<MetricAlarm>();
        var paginatedDescribeAlarms = _amazonCloudWatch.Paginators.DescribeAlarms(
            new DescribeAlarmsRequest()
            {
                StateValue = stateValue
            });

        await foreach (var data in paginatedDescribeAlarms.MetricAlarms)
        {
            alarms.Add(data);
        }
        return alarms;
    }

    /// <summary>
    /// Describe the current alarms for a specific metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarmsForMetric(string
metricNamespace, string metricName)
    {
        var alarmsResult = await _amazonCloudWatch.DescribeAlarmsForMetricAsync(
            new DescribeAlarmsForMetricRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName
            });

        return alarmsResult.MetricAlarms ?? new List<MetricAlarm>();
    }

    /// <summary>
    /// Describe the history of an alarm for a number of days in the past.
    /// </summary>
    /// <param name="alarmName">The name of the alarm.</param>
    /// <param name="historyDays">The number of days in the past.</param>
```

```
    /// <returns>The list of alarm history data.</returns>
    public async Task<List<AlarmHistoryItem>> DescribeAlarmHistory(string alarmName,
int historyDays)
    {
        List<AlarmHistoryItem> alarmHistory = new List<AlarmHistoryItem>();
        var paginatedAlarmHistory =
_amazonCloudWatch.Paginators.DescribeAlarmHistory(
            new DescribeAlarmHistoryRequest()
            {
                AlarmName = alarmName,
                EndDateUtc = DateTime.UtcNow,
                HistoryItemType = HistoryItemType.StateUpdate,
                StartDateUtc = DateTime.UtcNow.AddDays(-historyDays)
            });

        await foreach (var data in paginatedAlarmHistory.AlarmHistoryItems)
        {
            alarmHistory.Add(data);
        }
        return alarmHistory;
    }

    /// <summary>
    /// Delete a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteAlarms(List<string> alarmNames)
    {
        var deleteAlarmsResult = await _amazonCloudWatch.DeleteAlarmsAsync(
            new DeleteAlarmsRequest()
            {
                AlarmNames = alarmNames
            });

        return deleteAlarmsResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Disable the actions for a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DisableAlarmActions(List<string> alarmNames)
```

```
    {
        var disableAlarmActionsResult = await
_amazonCloudWatch.DisableAlarmActionsAsync(
            new DisableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });

        return disableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Enable the actions for a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> EnableAlarmActions(List<string> alarmNames)
    {
        var enableAlarmActionsResult = await
_amazonCloudWatch.EnableAlarmActionsAsync(
            new EnableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });

        return enableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Add an anomaly detector for a single metric.
    /// </summary>
    /// <param name="anomalyDetector">A single metric anomaly detector.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
    {
        var putAlarmDetectorResult = await
_amazonCloudWatch.PutAnomalyDetectorAsync(
            new PutAnomalyDetectorRequest()
            {
                SingleMetricAnomalyDetector = anomalyDetector
            });

        return putAlarmDetectorResult.HttpStatusCode == HttpStatusCode.OK;
```

```
    }

    /// <summary>
    /// Describe anomaly detectors for a metric and namespace.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The metric of the anomaly detectors.</param>
    /// <returns>The list of detectors.</returns>
    public async Task<List<AnomalyDetector>> DescribeAnomalyDetectors(string
metricNamespace, string metricName)
    {
        List<AnomalyDetector> detectors = new List<AnomalyDetector>();
        var paginatedDescribeAnomalyDetectors =
_amazonCloudWatch.Paginators.DescribeAnomalyDetectors(
            new DescribeAnomalyDetectorsRequest()
            {
                MetricName = metricName,
                Namespace = metricNamespace
            });

        await foreach (var data in
paginatedDescribeAnomalyDetectors.AnomalyDetectors)
        {
            detectors.Add(data);
        }

        return detectors;
    }

    /// <summary>
    /// Delete a single metric anomaly detector.
    /// </summary>
    /// <param name="anomalyDetector">The anomaly detector to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
    {
        var deleteAnomalyDetectorResponse = await
_amazonCloudWatch.DeleteAnomalyDetectorAsync(
            new DeleteAnomalyDetectorRequest()
            {
                SingleMetricAnomalyDetector = anomalyDetector
            });
```

```
        return deleteAnomalyDetectorResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete a list of CloudWatch dashboards.
    /// </summary>
    /// <param name="dashboardNames">List of dashboard names to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteDashboards(List<string> dashboardNames)
    {
        var deleteDashboardsResponse = await
 _amazonCloudWatch.DeleteDashboardsAsync(
            new DeleteDashboardsRequest()
            {
                DashboardNames = dashboardNames
            });

        return deleteDashboardsResponse.HttpStatusCode == HttpStatusCode.OK;
    }
}
```

Contoh nilai settings.json untuk skenario.

```
{
  "dashboardName": "example-new-dashboard",
  "exampleAlarmName": "example-metric-alarm",
  "accountId": "1234567890",
  "region": "us-east-1",
  "emailTopic": "Default_CloudWatch_Alarms_Topic",
  "customMetricNamespace": "example-namespace",
  "customMetricName": "example-custom-metric",
  "dashboardExampleBody": {
    "widgets": [
      {
        "height": 6,
        "width": 6,
        "y": 0,
        "x": 0,
        "type": "text",
        "properties": {
          "markdown": "# Code Example Dashboard \nThis dashboard was created by
 example code.\n"
```

```
        }
      },
      {
        "height": 8,
        "width": 8,
        "y": 0,
        "x": 6,
        "type": "metric",
        "properties": {
          "metrics": [
            [
              "AWS/Billing",
              "EstimatedCharges",
              "Currency",
              "USD",
              { "region": "us-east-1" }
            ]
          ],
          "view": "timeSeries",
          "region": "us-east-1",
          "stat": "Maximum",
          "period": 86400,
          "yAxis": {
            "left": {
              "min": 0,
              "max": 100
            }
          },
          "stacked": false,
          "title": "Estimated Billing",
          "setPeriodToTimeRange": false,
          "liveData": true,
          "sparkline": true,
          "trend": true
        }
      },
      {
        "height": 8,
        "width": 8,
        "y": 0,
        "x": 14,
        "type": "metric",
        "properties": {
          "metrics": [
```

```
            [ "AWS/Usage", "CallCount", "Type", "API", "Resource", "ListMetrics",
 "Service", "CloudWatch", "Class", "None" ],
            [ "...", "GetMetricStatistics", ".", ".", ".", "." ],
            [ "...", "GetMetricData", ".", ".", ".", "." ],
            [ "...", "PutDashboard", ".", ".", ".", "." ],
            [ "...", "PutMetricData", ".", ".", ".", "." ]
          ],
          "view": "timeSeries",
          "yAxis": {
            "left": {
              "min": 0,
              "max": 200
            }
          },
          "stacked": false,
          "region": "us-east-1",
          "stat": "Sum",
          "period": 300,
          "title": "CloudWatch Usage",
          "setPeriodToTimeRange": false,
          "liveData": true,
          "sparkline": true,
          "trend": true
        }
      }
    ]
  }
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk .NET .
  - DeleteAlarms
  - DeleteAnomalyDetector
  - DeleteDashboards
  - DescribeAlarmHistory
  - DescribeAlarms
  - DescribeAlarmsForMetric
  - DescribeAnomalyDetectors
  - GetMetricData

- GetMetricStatistics
- GetMetricWidgetImage
- ListMetrics
- PutAnomalyDetector
- PutDashboard
- PutMetricAlarm
- PutMetricData

Tindakan

`DeleteAlarms`

Contoh kode berikut menunjukkan cara menggunakan`DeleteAlarms`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
/// <summary>
/// Delete a list of alarms from CloudWatch.
/// </summary>
/// <param name="alarmNames">A list of names of alarms to delete.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteAlarms(List<string> alarmNames)
{
    var deleteAlarmsResult = await _amazonCloudWatch.DeleteAlarmsAsync(
        new DeleteAlarmsRequest()
        {
            AlarmNames = alarmNames
        });

    return deleteAlarmsResult.HttpStatusCode == HttpStatusCode.OK;
}
```

- Untuk detail API, lihat DeleteAlarmsdi Referensi AWS SDK untuk .NET API.

`DeleteAnomalyDetector`

Contoh kode berikut menunjukkan cara menggunakan`DeleteAnomalyDetector`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   /// <summary>
   /// Delete a single metric anomaly detector.
   /// </summary>
   /// <param name="anomalyDetector">The anomaly detector to delete.</param>
   /// <returns>True if successful.</returns>
   public async Task<bool> DeleteAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
   {
       var deleteAnomalyDetectorResponse = await
_amazonCloudWatch.DeleteAnomalyDetectorAsync(
           new DeleteAnomalyDetectorRequest()
           {
               SingleMetricAnomalyDetector = anomalyDetector
           });

       return deleteAnomalyDetectorResponse.HttpStatusCode == HttpStatusCode.OK;
   }
```

- Untuk detail API, lihat DeleteAnomalyDetectordi Referensi AWS SDK untuk .NET API.

`DeleteDashboards`

Contoh kode berikut menunjukkan cara menggunakan`DeleteDashboards`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Delete a list of CloudWatch dashboards.
    /// </summary>
    /// <param name="dashboardNames">List of dashboard names to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteDashboards(List<string> dashboardNames)
    {
        var deleteDashboardsResponse = await
_amazonCloudWatch.DeleteDashboardsAsync(
            new DeleteDashboardsRequest()
            {
                DashboardNames = dashboardNames
            });

        return deleteDashboardsResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- Untuk detail API, lihat DeleteDashboardsdi Referensi AWS SDK untuk .NET API.

`DescribeAlarmHistory`

Contoh kode berikut menunjukkan cara menggunakan`DescribeAlarmHistory`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Describe the history of an alarm for a number of days in the past.
    /// </summary>
    /// <param name="alarmName">The name of the alarm.</param>
    /// <param name="historyDays">The number of days in the past.</param>
    /// <returns>The list of alarm history data.</returns>
    public async Task<List<AlarmHistoryItem>> DescribeAlarmHistory(string alarmName,
int historyDays)
    {
        List<AlarmHistoryItem> alarmHistory = new List<AlarmHistoryItem>();
        var paginatedAlarmHistory =
_amazonCloudWatch.Paginators.DescribeAlarmHistory(
            new DescribeAlarmHistoryRequest()
            {
                AlarmName = alarmName,
                EndDateUtc = DateTime.UtcNow,
                HistoryItemType = HistoryItemType.StateUpdate,
                StartDateUtc = DateTime.UtcNow.AddDays(-historyDays)
            });

        await foreach (var data in paginatedAlarmHistory.AlarmHistoryItems)
        {
            alarmHistory.Add(data);
        }
        return alarmHistory;
    }
```

- Untuk detail API, lihat DescribeAlarmHistorydi Referensi AWS SDK untuk .NET API.

`DescribeAlarms`

Contoh kode berikut menunjukkan cara menggunakan`DescribeAlarms`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Describe the current alarms, optionally filtered by state.
    /// </summary>
    /// <param name="stateValue">Optional filter for alarm state.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarms(StateValue? stateValue =
null)
    {
        List<MetricAlarm> alarms = new List<MetricAlarm>();
        var paginatedDescribeAlarms = _amazonCloudWatch.Paginators.DescribeAlarms(
            new DescribeAlarmsRequest()
            {
                StateValue = stateValue
            });

        await foreach (var data in paginatedDescribeAlarms.MetricAlarms)
        {
            alarms.Add(data);
        }
        return alarms;
    }
```

- Untuk detail API, lihat DescribeAlarmsdi Referensi AWS SDK untuk .NET API.

`DescribeAlarmsForMetric`

Contoh kode berikut menunjukkan cara menggunakan`DescribeAlarmsForMetric`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Describe the current alarms for a specific metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
```

```
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarmsForMetric(string
metricNamespace, string metricName)
    {
        var alarmsResult = await _amazonCloudWatch.DescribeAlarmsForMetricAsync(
            new DescribeAlarmsForMetricRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName
            });

        return alarmsResult.MetricAlarms ?? new List<MetricAlarm>();
    }
```

- Untuk detail API, lihat DescribeAlarmsForMetricdi Referensi AWS SDK untuk .NET API.

`DescribeAnomalyDetectors`

Contoh kode berikut menunjukkan cara menggunakan`DescribeAnomalyDetectors`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Describe anomaly detectors for a metric and namespace.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The metric of the anomaly detectors.</param>
    /// <returns>The list of detectors.</returns>
    public async Task<List<AnomalyDetector>> DescribeAnomalyDetectors(string
metricNamespace, string metricName)
    {
        List<AnomalyDetector> detectors = new List<AnomalyDetector>();
        var paginatedDescribeAnomalyDetectors =
_amazonCloudWatch.Paginators.DescribeAnomalyDetectors(
```

```
            new DescribeAnomalyDetectorsRequest()
            {
                MetricName = metricName,
                Namespace = metricNamespace
            });

        await foreach (var data in
paginatedDescribeAnomalyDetectors.AnomalyDetectors)
        {
            detectors.Add(data);
        }

        return detectors;
    }
```

- Untuk detail API, lihat DescribeAnomalyDetectorsdi Referensi AWS SDK untuk .NET API.

`DisableAlarmActions`

Contoh kode berikut menunjukkan cara menggunakan`DisableAlarmActions`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Disable the actions for a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DisableAlarmActions(List<string> alarmNames)
    {
        var disableAlarmActionsResult = await
_amazonCloudWatch.DisableAlarmActionsAsync(
            new DisableAlarmActionsRequest()
            {
```

```
                AlarmNames = alarmNames
            });

        return disableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }
```

- Untuk detail API, lihat DisableAlarmActionsdi Referensi AWS SDK untuk .NET API.

`EnableAlarmActions`

Contoh kode berikut menunjukkan cara menggunakan`EnableAlarmActions`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Enable the actions for a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> EnableAlarmActions(List<string> alarmNames)
    {
        var enableAlarmActionsResult = await
_amazonCloudWatch.EnableAlarmActionsAsync(
            new EnableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });

        return enableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }
```

- Untuk detail API, lihat EnableAlarmActionsdi Referensi AWS SDK untuk .NET API.

`GetDashboard`

Contoh kode berikut menunjukkan cara menggunakan`GetDashboard`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
/// <summary>
/// Get information on a dashboard.
/// </summary>
/// <param name="dashboardName">The name of the dashboard.</param>
/// <returns>A JSON object with dashboard information.</returns>
public async Task<string> GetDashboard(string dashboardName)
{
    var dashboardResponse = await _amazonCloudWatch.GetDashboardAsync(
        new GetDashboardRequest()
        {
            DashboardName = dashboardName
        });

    return dashboardResponse.DashboardBody;
}
```

- Untuk detail API, lihat GetDashboarddi Referensi AWS SDK untuk .NET API.

`GetMetricData`

Contoh kode berikut menunjukkan cara menggunakan`GetMetricData`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Get data for CloudWatch metrics.
    /// </summary>
    /// <param name="minutesOfData">The number of minutes of data to include.</
param>
    /// <param name="useDescendingTime">True to return the data descending by
 time.</param>
    /// <param name="endDateUtc">The end date for the data, in UTC.</param>
    /// <param name="maxDataPoints">The maximum data points to include.</param>
    /// <param name="dataQueries">Optional data queries to include.</param>
    /// <returns>A list of the requested metric data.</returns>
    public async Task<List<MetricDataResult>> GetMetricData(int minutesOfData, bool
 useDescendingTime, DateTime? endDateUtc = null,
        int maxDataPoints = 0, List<MetricDataQuery>? dataQueries = null)
    {
        var metricData = new List<MetricDataResult>();
        // If no end time is provided, use the current time for the end time.
        endDateUtc ??= DateTime.UtcNow;
        var timeZoneOffset =
 TimeZoneInfo.Local.GetUtcOffset(endDateUtc.Value.ToLocalTime());
        var startTimeUtc = endDateUtc.Value.AddMinutes(-minutesOfData);
        // The timezone string should be in the format +0000, so use the timezone
 offset to format it correctly.
        var timeZoneString = $"{timeZoneOffset.Hours:D2}
{timeZoneOffset.Minutes:D2}";
        // Add the plus sign for positive offsets.
        timeZoneString = timeZoneString.StartsWith('-') ? timeZoneString : "+" +
 timeZoneString;
        var paginatedMetricData = _amazonCloudWatch.Paginators.GetMetricData(
            new GetMetricDataRequest()
            {
                StartTimeUtc = startTimeUtc,
                EndTimeUtc = endDateUtc.Value,
                LabelOptions = new LabelOptions { Timezone = timeZoneString },
```

```
                ScanBy = useDescendingTime ? ScanBy.TimestampDescending :
ScanBy.TimestampAscending,
                MaxDatapoints = maxDataPoints,
                MetricDataQueries = dataQueries,
            });

        if (paginatedMetricData.MetricDataResults != null)
        {
            await foreach (var data in paginatedMetricData.MetricDataResults)
            {
                metricData.Add(data);
            }
        }

        return metricData;
    }
```

- Untuk detail API, lihat GetMetricDatadi Referensi AWS SDK untuk .NET API.

`GetMetricStatistics`

Contoh kode berikut menunjukkan cara menggunakan`GetMetricStatistics`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Get billing statistics using a call to a wrapper class.
    /// </summary>
    /// <returns>A collection of billing statistics.</returns>
    private static async Task<List<Datapoint>> SetupBillingStatistics()
    {
        // Make a request for EstimatedCharges with a period of one day for the past
seven days.
        var billingStatistics = await _cloudWatchWrapper.GetMetricStatistics(
```

```
            "AWS/Billing",
            "EstimatedCharges",
            new List<string>() { "Maximum" },
            new List<Dimension>() { new Dimension { Name = "Currency", Value =
"USD" } },
            7,
            86400);

        billingStatistics = billingStatistics.OrderBy(n => n.Timestamp).ToList();

        return billingStatistics;
    }

    /// <summary>
    /// Wrapper to get statistics for a specific CloudWatch metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <param name="statistics">The list of statistics to include.</param>
    /// <param name="dimensions">The list of dimensions to include.</param>
    /// <param name="days">The number of days in the past to include.</param>
    /// <param name="period">The period for the data.</param>
    /// <returns>A list of DataPoint objects for the statistics.</returns>
    public async Task<List<Datapoint>> GetMetricStatistics(string metricNamespace,
        string metricName, List<string> statistics, List<Dimension> dimensions, int
days, int period)
    {
        var metricStatistics = await _amazonCloudWatch.GetMetricStatisticsAsync(
            new GetMetricStatisticsRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName,
                Dimensions = dimensions,
                Statistics = statistics,
                StartTimeUtc = DateTime.UtcNow.AddDays(-days),
                EndTimeUtc = DateTime.UtcNow,
                Period = period
            });

        return metricStatistics.Datapoints ?? new List<Datapoint>();
    }
```

- Untuk detail API, lihat GetMetricStatisticsdi Referensi AWS SDK untuk .NET API.

`GetMetricWidgetImage`

Contoh kode berikut menunjukkan cara menggunakan`GetMetricWidgetImage`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Get an image for a metric graphed over time.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metric">The name of the metric.</param>
    /// <param name="stat">The name of the stat to chart.</param>
    /// <param name="period">The period to use for the chart.</param>
    /// <returns>A memory stream for the chart image.</returns>
    public async Task<MemoryStream> GetTimeSeriesMetricImage(string metricNamespace,
string metric, string stat, int period)
    {
        var metricImageWidget = new
        {
            title = "Example Metric Graph",
            view = "timeSeries",
            stacked = false,
            period = period,
            width = 1400,
            height = 600,
            metrics = new List<List<object>>
                { new() { metricNamespace, metric, new { stat } } }
        };

        var metricImageWidgetString = JsonSerializer.Serialize(metricImageWidget);
        var imageResponse = await _amazonCloudWatch.GetMetricWidgetImageAsync(
            new GetMetricWidgetImageRequest()
            {
                MetricWidget = metricImageWidgetString
            });

        return imageResponse.MetricWidgetImage;
```

```
    }

    /// <summary>
    /// Save a metric image to a file.
    /// </summary>
    /// <param name="memoryStream">The MemoryStream for the metric image.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The path to the file.</returns>
    public string SaveMetricImage(MemoryStream memoryStream, string metricName)
    {
        var metricFileName = $"{metricName}_{DateTime.Now.Ticks}.png";
        using var sr = new StreamReader(memoryStream);
        // Writes the memory stream to a file.
        File.WriteAllBytes(metricFileName, memoryStream.ToArray());
        var filePath = Path.Join(AppDomain.CurrentDomain.BaseDirectory,
            metricFileName);
        return filePath;
    }
```

- Untuk detail API, lihat GetMetricWidgetImagedi Referensi AWS SDK untuk .NET API.

`ListDashboards`

Contoh kode berikut menunjukkan cara menggunakan`ListDashboards`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Get a list of dashboards.
    /// </summary>
    /// <returns>A list of DashboardEntry objects.</returns>
    public async Task<List<DashboardEntry>> ListDashboards()
    {
        var results = new List<DashboardEntry>();
```

```
        var paginateDashboards = _amazonCloudWatch.Paginators.ListDashboards(
            new ListDashboardsRequest());
        // Get the entire list using the paginator.
        await foreach (var data in paginateDashboards.DashboardEntries)
        {
            results.Add(data);
        }

        return results;
    }
```

- Untuk detail API, lihat ListDashboardsdi Referensi AWS SDK untuk .NET API.

`ListMetrics`

Contoh kode berikut menunjukkan cara menggunakan`ListMetrics`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// List metrics available, optionally within a namespace.
    /// </summary>
    /// <param name="metricNamespace">Optional CloudWatch namespace to use when
listing metrics.</param>
    /// <param name="filter">Optional dimension filter.</param>
    /// <param name="metricName">Optional metric name filter.</param>
    /// <returns>The list of metrics.</returns>
    public async Task<List<Metric>> ListMetrics(string? metricNamespace = null,
DimensionFilter? filter = null, string? metricName = null)
    {
        var results = new List<Metric>();
        var paginateMetrics = _amazonCloudWatch.Paginators.ListMetrics(
            new ListMetricsRequest
            {
                Namespace = metricNamespace,
```

```
                Dimensions = filter != null ? new List<DimensionFilter> { filter } :
null,
                MetricName = metricName
            });
        // Get the entire list using the paginator.
        await foreach (var metric in paginateMetrics.Metrics)
        {
            results.Add(metric);
        }

        return results;
    }
```

- Untuk detail API, lihat ListMetricsdi Referensi AWS SDK untuk .NET API.

`PutAnomalyDetector`

Contoh kode berikut menunjukkan cara menggunakan`PutAnomalyDetector`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Add an anomaly detector for a single metric.
    /// </summary>
    /// <param name="anomalyDetector">A single metric anomaly detector.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
    {
        var putAlarmDetectorResult = await
_amazonCloudWatch.PutAnomalyDetectorAsync(
            new PutAnomalyDetectorRequest()
            {
                SingleMetricAnomalyDetector = anomalyDetector
            });
```

```
        return putAlarmDetectorResult.HttpStatusCode == HttpStatusCode.OK;
    }
```

- Untuk detail API, lihat PutAnomalyDetectordi Referensi AWS SDK untuk .NET API.

`PutDashboard`

Contoh kode berikut menunjukkan cara menggunakan`PutDashboard`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
/// <summary>
/// Set up a dashboard using a call to the wrapper class.
/// </summary>
/// <param name="customMetricNamespace">The metric namespace.</param>
/// <param name="customMetricName">The metric name.</param>
/// <param name="dashboardName">The name of the dashboard.</param>
/// <returns>A list of validation messages.</returns>
private static async Task<List<DashboardValidationMessage>> SetupDashboard(
    string customMetricNamespace, string customMetricName, string dashboardName)
{
    // Get the dashboard model from configuration.
    var newDashboard = new DashboardModel();
    _configuration.GetSection("dashboardExampleBody").Bind(newDashboard);

    // Add a new metric to the dashboard.
    newDashboard.Widgets.Add(new Widget
    {
        Height = 8,
        Width = 8,
        Y = 8,
        X = 0,
        Type = "metric",
```

```
            Properties = new Properties
            {
                Metrics = new List<List<object>>
                    { new() { customMetricNamespace, customMetricName } },
                View = "timeSeries",
                Region = "us-east-1",
                Stat = "Sum",
                Period = 86400,
                YAxis = new YAxis { Left = new Left { Min = 0, Max = 100 } },
                Title = "Custom Metric Widget",
                LiveData = true,
                Sparkline = true,
                Trend = true,
                Stacked = false,
                SetPeriodToTimeRange = false
            }
        });

        var newDashboardString = JsonSerializer.Serialize(newDashboard,
            new JsonSerializerOptions
            { DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull });
        var validationMessages =
            await _cloudWatchWrapper.PutDashboard(dashboardName,
 newDashboardString);

        return validationMessages;
    }

    /// <summary>
    /// Wrapper to create or add to a dashboard with metrics.
    /// </summary>
    /// <param name="dashboardName">The name for the dashboard.</param>
    /// <param name="dashboardBody">The metric data in JSON for the dashboard.</
param>
    /// <returns>A list of validation messages for the dashboard.</returns>
    public async Task<List<DashboardValidationMessage>> PutDashboard(string
 dashboardName,
        string dashboardBody)
    {
        // Updating a dashboard replaces all contents.
        // Best practice is to include a text widget indicating this dashboard was
 created programmatically.
        var dashboardResponse = await _amazonCloudWatch.PutDashboardAsync(
            new PutDashboardRequest()
```

```
            {
                DashboardName = dashboardName,
                DashboardBody = dashboardBody
            });

        return dashboardResponse.DashboardValidationMessages ?? new
List<DashboardValidationMessage>();
    }
```

- Untuk detail API, lihat PutDashboarddi Referensi AWS SDK untuk .NET API.

`PutMetricAlarm`

Contoh kode berikut menunjukkan cara menggunakan`PutMetricAlarm`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Add a metric alarm to send an email when the metric passes a threshold.
    /// </summary>
    /// <param name="alarmDescription">A description of the alarm.</param>
    /// <param name="alarmName">The name for the alarm.</param>
    /// <param name="comparison">The type of comparison to use.</param>
    /// <param name="metricName">The name of the metric for the alarm.</param>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="threshold">The threshold value for the alarm.</param>
    /// <param name="alarmActions">Optional actions to execute when in an alarm
state.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutMetricEmailAlarm(string alarmDescription, string
alarmName, ComparisonOperator comparison,
        string metricName, string metricNamespace, double threshold, List<string>
alarmActions = null!)
    {
```

```
        try
        {
            var putEmailAlarmResponse = await _amazonCloudWatch.PutMetricAlarmAsync(
                new PutMetricAlarmRequest()
                {
                    AlarmActions = alarmActions,
                    AlarmDescription = alarmDescription,
                    AlarmName = alarmName,
                    ComparisonOperator = comparison,
                    Threshold = threshold,
                    Namespace = metricNamespace,
                    MetricName = metricName,
                    EvaluationPeriods = 1,
                    Period = 10,
                    Statistic = new Statistic("Maximum"),
                    DatapointsToAlarm = 1,
                    TreatMissingData = "ignore"
                });
            return putEmailAlarmResponse.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (LimitExceededException lex)
        {
            _logger.LogError(lex, $"Unable to add alarm {alarmName}. Alarm quota has
already been reached.");
        }

        return false;
    }

    /// <summary>
    /// Add specific email actions to a list of action strings for a CloudWatch
alarm.
    /// </summary>
    /// <param name="accountId">The AccountId for the alarm.</param>
    /// <param name="region">The region for the alarm.</param>
    /// <param name="emailTopicName">An Amazon Simple Notification Service (SNS)
topic for the alarm email.</param>
    /// <param name="alarmActions">Optional list of existing alarm actions to append
to.</param>
    /// <returns>A list of string actions for an alarm.</returns>
    public List<string> AddEmailAlarmAction(string accountId, string region,
        string emailTopicName, List<string>? alarmActions = null)
    {
        alarmActions ??= new List<string>();
```

```
        var snsAlarmAction = $"arn:aws:sns:{region}:{accountId}:{emailTopicName}";
        alarmActions.Add(snsAlarmAction);
        return alarmActions;
    }
```

- Untuk detail API, lihat PutMetricAlarmdi Referensi AWS SDK untuk .NET API.

`PutMetricData`

Contoh kode berikut menunjukkan cara menggunakan`PutMetricData`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Add some metric data using a call to a wrapper class.
    /// </summary>
    /// <param name="customMetricName">The metric name.</param>
    /// <param name="customMetricNamespace">The metric namespace.</param>
    /// <returns></returns>
    private static async Task<List<MetricDatum>> PutRandomMetricData(string
customMetricName,
        string customMetricNamespace)
    {
        List<MetricDatum> customData = new List<MetricDatum>();
        Random rnd = new Random();

        // Add 10 random values up to 100, starting with a timestamp 15 minutes in
the past.
        var utcNowMinus15 = DateTime.UtcNow.AddMinutes(-15);
        for (int i = 0; i < 10; i++)
        {
            var metricValue = rnd.Next(0, 100);
            customData.Add(
                new MetricDatum
```

```
                {
                    MetricName = customMetricName,
                    Value = metricValue,
                    TimestampUtc = utcNowMinus15.AddMinutes(i)
                }
            );
        }

        await _cloudWatchWrapper.PutMetricData(customMetricNamespace, customData);
        return customData;
    }

    /// <summary>
    /// Wrapper to add metric data to a CloudWatch metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricData">A data object for the metric data.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutMetricData(string metricNamespace,
        List<MetricDatum> metricData)
    {
        var putDataResponse = await _amazonCloudWatch.PutMetricDataAsync(
            new PutMetricDataRequest()
            {
                MetricData = metricData,
                Namespace = metricNamespace,
            });

        return putDataResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- Untuk detail API, lihat PutMetricDatadi Referensi AWS SDK untuk .NET API.

CloudWatch Log contoh menggunakan SDK untuk .NET (v4)

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK untuk .NET (v4) dengan CloudWatch Log.

Tindakan merupakan 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

`GetQueryResults`

Contoh kode berikut menunjukkan cara menggunakan`GetQueryResults`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
/// <summary>
/// Gets the results of a CloudWatch Logs Insights query.
/// </summary>
/// <param name="queryId">The ID of the query.</param>
/// <returns>The query results response.</returns>
public async Task<GetQueryResultsResponse?> GetQueryResultsAsync(string queryId)
{
    try
    {
        var request = new GetQueryResultsRequest
        {
            QueryId = queryId
        };
```

```
            var response = await
 _amazonCloudWatchLogs.GetQueryResultsAsync(request);
            return response;
        }
        catch (ResourceNotFoundException ex)
        {
            _logger.LogError($"Query not found: {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"An error occurred while getting query results:
 {ex.Message}");
            return null;
        }
    }
```

- Untuk detail API, lihat GetQueryResultsdi Referensi AWS SDK untuk .NET API.

`StartQuery`

Contoh kode berikut menunjukkan cara menggunakan`StartQuery`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Starts a CloudWatch Logs Insights query.
    /// </summary>
    /// <param name="logGroupName">The name of the log group to query.</param>
    /// <param name="queryString">The CloudWatch Logs Insights query string.</param>
    /// <param name="startTime">The start time for the query (seconds since
 epoch).</param>
    /// <param name="endTime">The end time for the query (seconds since epoch).</
param>
```

```
    /// <param name="limit">The maximum number of results to return.</param>
    /// <returns>The query ID if successful, null otherwise.</returns>
    public async Task<string?> StartQueryAsync(
        string logGroupName,
        string queryString,
        long startTime,
        long endTime,
        int limit = 10000)
    {
        try
        {
            var request = new StartQueryRequest
            {
                LogGroupName = logGroupName,
                QueryString = queryString,
                StartTime = startTime,
                EndTime = endTime,
                Limit = limit
            };

            var response = await _amazonCloudWatchLogs.StartQueryAsync(request);
            return response.QueryId;
        }
        catch (InvalidParameterException ex)
        {
            _logger.LogError($"Invalid parameter for query: {ex.Message}");
            return null;
        }
        catch (ResourceNotFoundException ex)
        {
            _logger.LogError($"Log group not found: {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"An error occurred while starting query:
{ex.Message}");
            return null;
        }
    }
```

- Untuk detail API, lihat StartQuerydi Referensi AWS SDK untuk .NET API.

Skenario

Jalankan kueri besar

Contoh kode berikut menunjukkan cara menggunakan CloudWatch Log untuk menanyakan lebih dari 10.000 catatan.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Ini adalah alur kerja utama yang menunjukkan skenario kueri besar.

```
using System.Diagnostics;
using System.Text.RegularExpressions;
using Amazon.CloudFormation;
using Amazon.CloudFormation.Model;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;
using CloudWatchLogsActions;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;

namespace CloudWatchLogsScenario;

public class LargeQueryWorkflow
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.
    This .NET code example performs the following tasks for the CloudWatch Logs
 Large Query workflow:

    1. Prepare the Application:
       - Prompt the user to deploy CloudFormation stack and generate sample logs.
       - Deploy the CloudFormation template for resource creation.
       - Generate 50,000 sample log entries using CloudWatch Logs API.
```

```
       - Wait 5 minutes for logs to be fully ingested.

    2. Execute Large Query:
       - Perform recursive queries to retrieve all logs using binary search.
       - Display progress for each query executed.
       - Show total execution time and logs found.

    3. Clean up:
       - Prompt the user to delete the CloudFormation stack and all resources.
       - Destroy the CloudFormation stack and wait until removed.
    */

    public static ILogger<LargeQueryWorkflow> _logger = null!;
    public static CloudWatchLogsWrapper _wrapper = null!;
    public static IAmazonCloudFormation _amazonCloudFormation = null!;

    private static string _logGroupName = "/workflows/cloudwatch-logs/large-query";
    private static string _logStreamName = "stream1";
    private static long _queryStartDate;
    private static long _queryEndDate;

    public static bool _interactive = true;
    public static string _stackName = "CloudWatchLargeQueryStack";
    private static string _stackResourcePath = "../../../../../../../scenarios/
features/cloudwatch_logs_large_query/resources/stack.yaml";

    public static async Task Main(string[] args)
    {
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter("Microsoft", LogLevel.Information))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonCloudWatchLogs>()
                    .AddAWSService<IAmazonCloudFormation>()
                    .AddTransient<CloudWatchLogsWrapper>()
            )
            .Build();

        if (_interactive)
        {
            _logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
                .CreateLogger<LargeQueryWorkflow>();
```

```
            _wrapper = host.Services.GetRequiredService<CloudWatchLogsWrapper>();
            _amazonCloudFormation =
host.Services.GetRequiredService<IAmazonCloudFormation>();
        }

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the CloudWatch Logs Large Query Scenario.");
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("This scenario demonstrates how to perform large-scale
queries on");
        Console.WriteLine("CloudWatch Logs using recursive binary search to retrieve
more than");
        Console.WriteLine("the 10,000 result limit.");
        Console.WriteLine();

        try
        {
            Console.WriteLine(new string('-', 80));
            var prepareSuccess = await PrepareApplication();
            Console.WriteLine(new string('-', 80));

            if (prepareSuccess)
            {
                Console.WriteLine(new string('-', 80));
                await ExecuteLargeQuery();
                Console.WriteLine(new string('-', 80));
            }

            Console.WriteLine(new string('-', 80));
            await Cleanup();
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "There was a problem with the scenario, initiating
cleanup...");
            _interactive = false;
            await Cleanup();
        }

        Console.WriteLine("CloudWatch Logs Large Query scenario completed.");
    }

    /// <summary>
```

```
    /// Runs the scenario workflow. Used for testing.
    /// </summary>
    public static async Task RunScenario()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the CloudWatch Logs Large Query Scenario.");
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("This scenario demonstrates how to perform large-scale
queries on");
        Console.WriteLine("CloudWatch Logs using recursive binary search to retrieve
more than");
        Console.WriteLine("the 10,000 result limit.");
        Console.WriteLine();

        try
        {
            Console.WriteLine(new string('-', 80));
            var prepareSuccess = await PrepareApplication();
            Console.WriteLine(new string('-', 80));

            if (prepareSuccess)
            {
                Console.WriteLine(new string('-', 80));
                await ExecuteLargeQuery();
                Console.WriteLine(new string('-', 80));
            }

            Console.WriteLine(new string('-', 80));
            await Cleanup();
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "There was a problem with the scenario, initiating
cleanup...");
            _interactive = false;
            await Cleanup();
        }

        Console.WriteLine("CloudWatch Logs Large Query scenario completed.");
    }

    /// <summary>
    /// Prepares the application by creating the necessary resources.
```

```
    /// </summary>
    /// <returns>True if the application was prepared successfully.</returns>
    public static async Task<bool> PrepareApplication()
    {
        Console.WriteLine("Preparing the application...");
        Console.WriteLine();

        try
        {
            var deployStack = !_interactive || GetYesNoResponse(
                "Would you like to deploy the CloudFormation stack and generate
sample logs? (y/n) ");

            if (deployStack)
            {
                if (_interactive)
                {
                    Console.Write(
                        $"Enter a path for the CloudFormation stack resource .yaml
file (or press Enter for default '{_stackResourcePath}'): ");
                    string? inputPath = Console.ReadLine();
                    if (!string.IsNullOrWhiteSpace(inputPath))
                    {
                        _stackResourcePath = inputPath;
                    }
                }

                _stackName = PromptUserForStackName();

                var deploySuccess = await DeployCloudFormationStack(_stackName);

                if (deploySuccess)
                {
                    Console.WriteLine();
                    Console.WriteLine("Generating 50,000 sample log entries...");
                    var generateSuccess = await GenerateSampleLogs();

                    if (generateSuccess)
                    {
                        Console.WriteLine();
                        Console.WriteLine("Sample logs created. Waiting 5 minutes
for logs to be fully ingested...");
                        await WaitWithCountdown(300);
```

```
                        Console.WriteLine("Application preparation complete.");
                        return true;
                    }
                }
            }
            else
            {
                _logGroupName = PromptUserForInput("Enter the log group name ",
_logGroupName);
                _logStreamName = PromptUserForInput("Enter the log stream name ",
_logStreamName);

                var startDateMs = PromptUserForLong("Enter the query start date
(milliseconds since epoch): ");
                var endDateMs = PromptUserForLong("Enter the query end date
(milliseconds since epoch): ");

                _queryStartDate = startDateMs / 1000;
                _queryEndDate = endDateMs / 1000;

                Console.WriteLine("Application preparation complete.");
                return true;
            }
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "An error occurred while preparing the
application.");
        }

        Console.WriteLine("Application preparation failed.");
        return false;
    }

    /// <summary>
    /// Deploys the CloudFormation stack with the necessary resources.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <returns>True if the stack was deployed successfully.</returns>
    private static async Task<bool> DeployCloudFormationStack(string stackName)
    {
        Console.WriteLine($"\nDeploying CloudFormation stack: {stackName}");

        try
```

```
        {
            var request = new CreateStackRequest
            {
                StackName = stackName,
                TemplateBody = await File.ReadAllTextAsync(_stackResourcePath)
            };

            var response = await _amazonCloudFormation.CreateStackAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"CloudFormation stack creation started:
{stackName}");

                bool stackCreated = await WaitForStackCompletion(response.StackId);

                if (stackCreated)
                {
                    Console.WriteLine("CloudFormation stack created successfully.");
                    return true;
                }
                else
                {
                    _logger.LogError($"CloudFormation stack creation failed:
{stackName}");
                    return false;
                }
            }
            else
            {
                _logger.LogError($"Failed to create CloudFormation stack:
{stackName}");
                return false;
            }
        }
        catch (AlreadyExistsException)
        {
            _logger.LogWarning($"CloudFormation stack '{stackName}' already exists.
Please provide a unique name.");
            var newStackName = PromptUserForStackName();
            return await DeployCloudFormationStack(newStackName);
        }
        catch (Exception ex)
        {
```

```
            _logger.LogError(ex, $"An error occurred while deploying the
CloudFormation stack: {stackName}");
            return false;
        }
    }

    /// <summary>
    /// Waits for the CloudFormation stack to be in the CREATE_COMPLETE state.
    /// </summary>
    /// <param name="stackId">The ID of the CloudFormation stack.</param>
    /// <returns>True if the stack was created successfully.</returns>
    private static async Task<bool> WaitForStackCompletion(string stackId)
    {
        int retryCount = 0;
        const int maxRetries = 30;
        const int retryDelay = 10000;

        while (retryCount < maxRetries)
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackId
            };

            var describeStacksResponse = await
_amazonCloudFormation.DescribeStacksAsync(describeStacksRequest);

            if (describeStacksResponse.Stacks.Count > 0)
            {
                if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_COMPLETE)
                {
                    return true;
                }
                if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_FAILED ||
                    describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.ROLLBACK_COMPLETE)
                {
                    return false;
                }
            }
```

```
            Console.WriteLine("Waiting for CloudFormation stack creation to
complete...");
            await Task.Delay(retryDelay);
            retryCount++;
        }

        _logger.LogError("Timed out waiting for CloudFormation stack creation to
complete.");
        return false;
    }

    /// <summary>
    /// Generates sample logs directly using CloudWatch Logs API.
    /// Creates 50,000 log entries spanning 5 minutes.
    /// </summary>
    /// <returns>True if logs were generated successfully.</returns>
    private static async Task<bool> GenerateSampleLogs()
    {
        const int totalEntries = 50000;
        const int entriesPerBatch = 10000;
        const int fiveMinutesMs = 5 * 60 * 1000;

        try
        {
            // Calculate timestamps
            var startTimeMs = DateTimeOffset.UtcNow.ToUnixTimeMilliseconds();
            var timestampIncrement = fiveMinutesMs / totalEntries;

            Console.WriteLine($"Generating {totalEntries} log entries...");

            var entryCount = 0;
            var currentTimestamp = startTimeMs;
            var numBatches = totalEntries / entriesPerBatch;

            // Generate and upload logs in batches
            for (int batchNum = 0; batchNum < numBatches; batchNum++)
            {
                var logEvents = new List<InputLogEvent>();

                for (int i = 0; i < entriesPerBatch; i++)
                {
                    logEvents.Add(new InputLogEvent
                    {
```

```
                        Timestamp =
 DateTimeOffset.FromUnixTimeMilliseconds(currentTimestamp).UtcDateTime,
                        Message = $"Entry {entryCount}"
                    });

                    entryCount++;
                    currentTimestamp += timestampIncrement;
                }

                // Upload batch
                var success = await _wrapper.PutLogEventsAsync(_logGroupName,
 _logStreamName, logEvents);
                if (!success)
                {
                    _logger.LogError($"Failed to upload batch {batchNum + 1}/
{numBatches}");
                    return false;
                }

                Console.WriteLine($"Uploaded batch {batchNum + 1}/{numBatches}");
            }

            // Set query date range (convert milliseconds to seconds for query API)
            _queryStartDate = startTimeMs / 1000;
            _queryEndDate = (currentTimestamp - timestampIncrement) / 1000;

            Console.WriteLine($"Query start date:
 {DateTimeOffset.FromUnixTimeSeconds(_queryStartDate):yyyy-MM-ddTHH:mm:ss.fffZ}");
            Console.WriteLine($"Query end date:
 {DateTimeOffset.FromUnixTimeSeconds(_queryEndDate):yyyy-MM-ddTHH:mm:ss.fffZ}");
            Console.WriteLine($"Successfully uploaded {totalEntries} log entries");

            return true;
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "An error occurred while generating sample logs.");
            return false;
        }
    }

    /// <summary>
    /// Executes the large query workflow.
    /// </summary>
```

```
    public static async Task ExecuteLargeQuery()
    {
        Console.WriteLine("Starting recursive query to retrieve all logs...");
        Console.WriteLine();

        var queryLimit = PromptUserForInteger("Enter the query limit (max 10000) ",
10000);
        if (queryLimit > 10000) queryLimit = 10000;

        var queryString = "fields @timestamp, @message | sort @timestamp asc";

        var stopwatch = Stopwatch.StartNew();
        var allResults = await PerformLargeQuery(_logGroupName, queryString,
_queryStartDate, _queryEndDate, queryLimit);
        stopwatch.Stop();

        Console.WriteLine();
        Console.WriteLine($"Queries finished in {stopwatch.Elapsed.TotalSeconds:F3}
seconds.");
        Console.WriteLine($"Total logs found: {allResults.Count}");

        // Check for duplicates
        Console.WriteLine();
        Console.WriteLine("Checking for duplicate logs...");
        var duplicates = FindDuplicateLogs(allResults);
        if (duplicates.Count > 0)
        {
            Console.WriteLine($"WARNING: Found {duplicates.Count} duplicate log
entries!");
            Console.WriteLine("Duplicate entries (showing first 10):");
            foreach (var dup in duplicates.Take(10))
            {
                Console.WriteLine($"  [{dup.Timestamp}] {dup.Message} (appears
{dup.Count} times)");
            }

            var uniqueCount = allResults.Count - duplicates.Sum(d => d.Count - 1);
            Console.WriteLine($"Unique logs: {uniqueCount}");
        }
        else
        {
            Console.WriteLine("No duplicates found. All logs are unique.");
        }
        Console.WriteLine();
```

```
        var viewSample = !_interactive || GetYesNoResponse("Would you like to see a
sample of the logs? (y/n) ");
        if (viewSample)
        {
            Console.WriteLine();
            Console.WriteLine($"Sample logs (first 10 of {allResults.Count}):");
            for (int i = 0; i < Math.Min(10, allResults.Count); i++)
            {
                var timestamp = allResults[i].Find(f => f.Field ==
"@timestamp")?.Value ?? "N/A";
                var message = allResults[i].Find(f => f.Field ==
"@message")?.Value ?? "N/A";
                Console.WriteLine($"[{timestamp}] {message}");
            }
        }
    }

    /// <summary>
    /// Performs a large query using recursive binary search.
    /// </summary>
    private static async Task<List<List<ResultField>>> PerformLargeQuery(
        string logGroupName,
        string queryString,
        long startTime,
        long endTime,
        int limit)
    {
        var queryId = await _wrapper.StartQueryAsync(logGroupName, queryString,
startTime, endTime, limit);
        if (queryId == null)
        {
            return new List<List<ResultField>>();
        }

        var results = await PollQueryResults(queryId);
        if (results == null || results.Count == 0)
        {
            return new List<List<ResultField>>();
        }

        var startDate =
DateTimeOffset.FromUnixTimeSeconds(startTime).ToString("yyyy-MM-ddTHH:mm:ss.fffZ");
```

```
        var endDate = DateTimeOffset.FromUnixTimeSeconds(endTime).ToString("yyyy-MM-
ddTHH:mm:ss.fffZ");
        Console.WriteLine($"Query date range: {startDate} ({startTime}s) to
 {endDate} ({endTime}s). Found {results.Count} logs.");

        if (results.Count < limit)
        {
            Console.WriteLine($"  -> Returning {results.Count} logs (less than limit
 of {limit})");
            return results;
        }

        Console.WriteLine($"  -> Hit limit of {limit}. Need to split and recurse.");

        // Get the timestamp of the last log (sorted to find the actual last one)
        var lastLogTimestamp = GetLastLogTimestamp(results);
        if (lastLogTimestamp == null)
        {
            Console.WriteLine($"  -> No timestamp found in results. Returning
 {results.Count} logs.");
            return results;
        }

        Console.WriteLine($"  -> Last log timestamp: {lastLogTimestamp}");

        // Parse the timestamp and add 1 millisecond to avoid querying the same log
 again
        var lastLogDate = DateTimeOffset.Parse(lastLogTimestamp + " +0000");
        Console.WriteLine($"  -> Last log as DateTimeOffset: {lastLogDate:yyyy-MM-
ddTHH:mm:ss.fffZ} ({lastLogDate.ToUnixTimeSeconds()}s)");

        var offsetLastLogDate = lastLogDate.AddMilliseconds(1);
        Console.WriteLine($"  -> Offset timestamp (last
 + 1ms): {offsetLastLogDate:yyyy-MM-ddTHH:mm:ss.fffZ}
 ({offsetLastLogDate.ToUnixTimeSeconds()}s)");

        // Convert to seconds, but round UP to the next second to avoid overlapping
 with logs in the same second
        // This ensures we don't re-query logs that share the same second as the
 last log
        var offsetLastLogTime = offsetLastLogDate.ToUnixTimeSeconds();
        if (offsetLastLogDate.Millisecond > 0)
        {
            offsetLastLogTime++; // Move to the next full second
```

```
            Console.WriteLine($"  -> Adjusted to next full second:
 {offsetLastLogTime}s ({DateTimeOffset.FromUnixTimeSeconds(offsetLastLogTime):yyyy-
MM-ddTHH:mm:ss.fffZ})");
        }

        Console.WriteLine($"  -> Comparing: offsetLastLogTime={offsetLastLogTime}s
 vs endTime={endTime}s");
        Console.WriteLine($"  -> End time as date:
 {DateTimeOffset.FromUnixTimeSeconds(endTime):yyyy-MM-ddTHH:mm:ss.fffZ}");

        // Check if there's any time range left to query
        if (offsetLastLogTime >= endTime)
        {
            Console.WriteLine($"  -> No time range left to query. Offset time
 ({offsetLastLogTime}s) >= end time ({endTime}s)");
            return results;
        }

        // Split the remaining date range in half
        var (range1Start, range1End, range2Start, range2End) =
 SplitDateRange(offsetLastLogTime, endTime);

        var range1StartDate =
 DateTimeOffset.FromUnixTimeSeconds(range1Start).ToString("yyyy-MM-
ddTHH:mm:ss.fffZ");
        var range1EndDate =
 DateTimeOffset.FromUnixTimeSeconds(range1End).ToString("yyyy-MM-ddTHH:mm:ss.fffZ");
        var range2StartDate =
 DateTimeOffset.FromUnixTimeSeconds(range2Start).ToString("yyyy-MM-
ddTHH:mm:ss.fffZ");
        var range2EndDate =
 DateTimeOffset.FromUnixTimeSeconds(range2End).ToString("yyyy-MM-ddTHH:mm:ss.fffZ");

        Console.WriteLine($"  -> Splitting remaining range:");
        Console.WriteLine($"     Range 1: {range1StartDate} ({range1Start}s) to
 {range1EndDate} ({range1End}s)");
        Console.WriteLine($"     Range 2: {range2StartDate} ({range2Start}s) to
 {range2EndDate} ({range2End}s)");

        // Query both halves recursively
        Console.WriteLine($"  -> Querying range 1...");
        var results1 = await PerformLargeQuery(logGroupName, queryString,
 range1Start, range1End, limit);
        Console.WriteLine($"  -> Range 1 returned {results1.Count} logs");
```

```
        Console.WriteLine($"  -> Querying range 2...");
        var results2 = await PerformLargeQuery(logGroupName, queryString,
range2Start, range2End, limit);
        Console.WriteLine($"  -> Range 2 returned {results2.Count} logs");

        // Combine all results
        var allResults = new List<List<ResultField>>(results);
        allResults.AddRange(results1);
        allResults.AddRange(results2);

        Console.WriteLine($"  -> Combined total: {allResults.Count} logs
({results.Count} + {results1.Count} + {results2.Count})");

        return allResults;
    }

    /// <summary>
    /// Gets the timestamp string of the most recent log from a list of logs.
    /// Sorts timestamps to find the actual last one.
    /// </summary>
    private static string? GetLastLogTimestamp(List<List<ResultField>> logs)
    {
        var timestamps = logs
            .Select(log => log.Find(f => f.Field == "@timestamp")?.Value)
            .Where(t => !string.IsNullOrEmpty(t))
            .OrderBy(t => t)
            .ToList();

        if (timestamps.Count == 0)
        {
            return null;
        }

        return timestamps[timestamps.Count - 1];
    }

    /// <summary>
    /// Splits a date range in half.
    /// Range 2 starts at midpoint + 1 second to avoid overlap.
    /// </summary>
    private static (long range1Start, long range1End, long range2Start, long
range2End) SplitDateRange(long startTime, long endTime)
    {
```

```
        var midpoint = startTime + (endTime - startTime) / 2;
        // Range 2 starts at midpoint + 1 to avoid querying the same second twice
        return (startTime, midpoint, midpoint + 1, endTime);
    }

    /// <summary>
    /// Polls for query results until complete.
    /// </summary>
    private static async Task<List<List<ResultField>>?> PollQueryResults(string
queryId)
    {
        int retryCount = 0;
        const int maxRetries = 60;
        const int retryDelay = 1000;

        while (retryCount < maxRetries)
        {
            var response = await _wrapper.GetQueryResultsAsync(queryId);
            if (response == null)
            {
                return null;
            }

            if (response.Status == QueryStatus.Complete)
            {
                return response.Results;
            }

            if (response.Status == QueryStatus.Failed ||
                response.Status == QueryStatus.Cancelled ||
                response.Status == QueryStatus.Timeout ||
                response.Status == QueryStatus.Unknown)
            {
                _logger.LogError($"Query failed with status: {response.Status}");
                return null;
            }

            await Task.Delay(retryDelay);
            retryCount++;
        }

        _logger.LogError("Timed out waiting for query results.");
        return null;
    }
```

```
   /// <summary>
   /// Cleans up the resources created during the scenario.
   /// </summary>
   public static async Task<bool> Cleanup()
   {
       var cleanup = !_interactive || GetYesNoResponse(
           "Do you want to delete the CloudFormation stack and all resources? (y/n)
");

       if (cleanup)
       {
           try
           {
               var stackDeleteSuccess = await DeleteCloudFormationStack(_stackName,
false);
               return stackDeleteSuccess;
           }
           catch (Exception ex)
           {
               _logger.LogError(ex, "An error occurred while cleaning up the
resources.");
               return false;
           }
       }

       Console.WriteLine($"Resources will remain. Stack name: {_stackName}, Log
group: {_logGroupName}");
       _logger.LogInformation("CloudWatch Logs Large Query scenario is complete.");
       return true;
   }

   /// <summary>
   /// Deletes the CloudFormation stack and waits for confirmation.
   /// </summary>
   private static async Task<bool> DeleteCloudFormationStack(string stackName, bool
forceDelete)
   {
       var request = new DeleteStackRequest
       {
           StackName = stackName,
       };

       if (forceDelete)
```

```
        {
            request.DeletionMode = DeletionMode.FORCE_DELETE_STACK;
        }

        await _amazonCloudFormation.DeleteStackAsync(request);
        Console.WriteLine($"CloudFormation stack '{stackName}' is being deleted.
This may take a few minutes.");

        bool stackDeleted = await WaitForStackDeletion(stackName, forceDelete);

        if (stackDeleted)
        {
            Console.WriteLine($"CloudFormation stack '{stackName}' has been
deleted.");
            return true;
        }
        else
        {
            _logger.LogError($"Failed to delete CloudFormation stack
'{stackName}'.");
            return false;
        }
    }

    /// <summary>
    /// Waits for the stack to be deleted.
    /// </summary>
    private static async Task<bool> WaitForStackDeletion(string stackName, bool
forceDelete)
    {
        int retryCount = 0;
        const int maxRetries = 30;
        const int retryDelay = 10000;

        while (retryCount < maxRetries)
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackName
            };

            try
            {
```

```
                var describeStacksResponse = await
_amazonCloudFormation.DescribeStacksAsync(describeStacksRequest);

                if (describeStacksResponse.Stacks.Count == 0 ||
                    describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.DELETE_COMPLETE)
                {
                    return true;
                }

                if (!forceDelete && describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.DELETE_FAILED)
                {
                    return await DeleteCloudFormationStack(stackName, true);
                }
            }
            catch (AmazonCloudFormationException ex) when (ex.ErrorCode ==
"ValidationError")
            {
                return true;
            }

            Console.WriteLine($"Waiting for CloudFormation stack '{stackName}' to be
deleted...");
            await Task.Delay(retryDelay);
            retryCount++;
        }

        _logger.LogError($"Timed out waiting for CloudFormation stack '{stackName}'
to be deleted.");
        return false;
    }

    /// <summary>
    /// Waits with a countdown display.
    /// </summary>
    private static async Task WaitWithCountdown(int seconds)
    {
        for (int i = seconds; i > 0; i--)
        {
            Console.Write($"\rWaiting: {i} seconds remaining...  ");
            await Task.Delay(1000);
        }
        Console.WriteLine("\rWait complete.                      ");
```

```
    }

    /// <summary>
    /// Helper method to get a yes or no response from the user.
    /// </summary>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null && ynResponse.Equals("y",
StringComparison.InvariantCultureIgnoreCase);
        return response;
    }

    /// <summary>
    /// Prompts the user for a stack name.
    /// </summary>
    private static string PromptUserForStackName()
    {
        if (_interactive)
        {
            Console.Write($"Enter a name for the CloudFormation stack (press Enter
for default '{_stackName}'): ");
            string? input = Console.ReadLine();
            if (!string.IsNullOrWhiteSpace(input))
            {
                var regex = "[a-zA-Z][-a-zA-Z0-9]*";
                if (!Regex.IsMatch(input, regex))
                {
                    Console.WriteLine($"Invalid stack name. Using default:
{_stackName}");
                    return _stackName;
                }
                return input;
            }
        }
        return _stackName;
    }

    /// <summary>
    /// Prompts the user for input with a default value.
    /// </summary>
    private static string PromptUserForInput(string prompt, string defaultValue)
    {
```

```
        if (_interactive)
        {
            Console.Write($"{prompt}(press Enter for default '{defaultValue}'): ");
            string? input = Console.ReadLine();
            return string.IsNullOrWhiteSpace(input) ? defaultValue : input;
        }
        return defaultValue;
    }

    /// <summary>
    /// Prompts the user for an integer value.
    /// </summary>
    private static int PromptUserForInteger(string prompt, int defaultValue)
    {
        if (_interactive)
        {
            Console.Write($"{prompt}(press Enter for default '{defaultValue}'): ");
            string? input = Console.ReadLine();
            if (string.IsNullOrWhiteSpace(input) || !int.TryParse(input, out var
result))
            {
                return defaultValue;
            }
            return result;
        }
        return defaultValue;
    }

    /// <summary>
    /// Prompts the user for a long value.
    /// </summary>
    private static long PromptUserForLong(string prompt)
    {
        if (_interactive)
        {
            Console.Write(prompt);
            string? input = Console.ReadLine();
            if (long.TryParse(input, out var result))
            {
                return result;
            }
        }
        return 0;
    }
```

```
    /// <summary>
    /// Finds duplicate log entries based on timestamp and message.
    /// </summary>
    private static List<(string Timestamp, string Message, int Count)>
 FindDuplicateLogs(List<List<ResultField>> logs)
    {
        var logSignatures = new Dictionary<string, int>();

        foreach (var log in logs)
        {
            var timestamp = log.Find(f => f.Field == "@timestamp")?.Value ?? "";
            var message = log.Find(f => f.Field == "@message")?.Value ?? "";
            var signature = $"{timestamp}|{message}";

            if (logSignatures.ContainsKey(signature))
            {
                logSignatures[signature]++;
            }
            else
            {
                logSignatures[signature] = 1;
            }
        }

        return logSignatures
            .Where(kvp => kvp.Value > 1)
            .Select(kvp =>
            {
                var parts = kvp.Key.Split('|');
                return (Timestamp: parts[0], Message: parts[1], Count: kvp.Value);
            })
            .OrderByDescending(x => x.Count)
            .ToList();
    }
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk .NET .
  - GetQueryResults
  - StartQuery

Contoh Penyedia Identitas Amazon Cognito menggunakan SDK untuk .NET (v4)

Contoh kode berikut menunjukkan cara melakukan tindakan dan menerapkan skenario umum dengan menggunakan AWS SDK untuk .NET (v4) dengan Penyedia Identitas Amazon Cognito.

Tindakan merupakan 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`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
/// <summary>
/// List the Amazon Cognito user pools for an account.
/// </summary>
/// <returns>A list of UserPoolDescriptionType objects.</returns>
public async Task<List<UserPoolDescriptionType>> ListUserPoolsAsync()
{
    var userPools = new List<UserPoolDescriptionType>();
```

```
        var userPoolsPaginator = _cognitoService.Paginators.ListUserPools(new
ListUserPoolsRequest());

        await foreach (var response in userPoolsPaginator.Responses)
        {
            userPools.AddRange(response.UserPools);
        }

        return userPools;
    }
```

- Untuk detail API, lihat ListUserPoolsdi Referensi AWS SDK untuk .NET API.

AWS Control Tower contoh menggunakan SDK untuk .NET (v4)

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK untuk .NET (v4) dengan AWS Control Tower.

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

Tindakan merupakan 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

- Memulai
- Hal-hal mendasar
- Tindakan

Memulai

Halo AWS Control Tower

Contoh kode berikut menunjukkan bagaimana untuk mulai menggunakan AWS Control Tower.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
using Amazon.ControlTower;
using Amazon.ControlTower.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;
using LogLevel = Microsoft.Extensions.Logging.LogLevel;

namespace ControlTowerActions;

/// <summary>
/// A class that introduces the AWS Control Tower by listing the
/// available baselines for the account.
/// </summary>
public class HelloControlTower
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for AWS Control Tower.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonControlTower>()
            )
            .Build();
```

```
        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<HelloControlTower>();

        var amazonClient = host.Services.GetRequiredService<IAmazonControlTower>();

        Console.Clear();
        Console.WriteLine("Hello, AWS Control Tower! Let's list available
 baselines:");
        Console.WriteLine();

        var baselines = new List<BaselineSummary>();

        try
        {
            var baselinesPaginator = amazonClient.Paginators.ListBaselines(new
 ListBaselinesRequest());

            await foreach (var response in baselinesPaginator.Responses)
            {
                baselines.AddRange(response.Baselines);
            }

            Console.WriteLine($"{baselines.Count} baseline(s) retrieved.");
            foreach (var baseline in baselines)
            {
                Console.WriteLine($"\t{baseline.Name}");
            }
        }
        catch (Amazon.ControlTower.Model.AccessDeniedException)
        {
            Console.WriteLine("Access denied. Please ensure you have the necessary
 permissions.");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred: {ex.Message}");
        }
    }
}
```

- Untuk detail API, lihat ListBaselinesdi Referensi AWS SDK untuk .NET API.

Hal-hal mendasar

Pelajari dasar-dasarnya

Contoh kode berikut ini menunjukkan cara untuk melakukan:

- Daftar zona pendaratan.
- Buat daftar, aktifkan, dapatkan, atur ulang, dan nonaktifkan garis dasar.
- Daftar, aktifkan, dapatkan, dan nonaktifkan kontrol.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Jalankan skenario interaktif yang menunjukkan AWS Control Tower fitur.

```
using Amazon.ControlCatalog;
using Amazon.ControlTower;
using Amazon.ControlTower.Model;
using Amazon.Organizations;
using Amazon.Organizations.Model;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;
using ControlTowerActions;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;

namespace ControlTowerBasics;

/// <summary>
/// Scenario class for AWS Control Tower basics.
/// </summary>
public class ControlTowerBasics
{
    public static bool isInteractive = true;
```

```
    public static ILogger logger = null!;
    public static IAmazonOrganizations? orgClient = null;
    public static IAmazonSecurityTokenService? stsClient = null;
    public static ControlTowerWrapper? wrapper = null;
    private static string? ouArn;
    private static bool useLandingZone = false;

    /// <summary>
    /// Main entry point for the AWS Control Tower basics scenario.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    public static async Task Main(string[] args)
    {
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonControlTower>()
                .AddAWSService<IAmazonControlCatalog>()
                .AddAWSService<IAmazonOrganizations>()
                .AddAWSService<IAmazonSecurityTokenService>()
                .AddTransient<ControlTowerWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<ControlTowerBasics>();

        wrapper = host.Services.GetRequiredService<ControlTowerWrapper>();
        orgClient = host.Services.GetRequiredService<IAmazonOrganizations>();
        stsClient = host.Services.GetRequiredService<IAmazonSecurityTokenService>();

        await RunScenario();
    }

    /// <summary>
    /// Runs the example scenario.
    /// </summary>
    public static async Task RunScenario()
    {
        Console.WriteLine(new string('-', 88));
        Console.WriteLine("\tWelcome to the AWS Control Tower with ControlCatalog
example scenario.");
        Console.WriteLine(new string('-', 88));
        Console.WriteLine("This demo will walk you through working with AWS Control
Tower for landing zones,");
```

```
        Console.WriteLine("managing baselines, and working with controls.");

        try
        {
            var accountId = (await stsClient!.GetCallerIdentityAsync(new
 GetCallerIdentityRequest())).Account;
            Console.WriteLine($"\nAccount ID: {accountId}");

            Console.WriteLine("\nSome demo operations require the use of a landing
 zone.");
            Console.WriteLine("You can use an existing landing zone or opt out of
 these operations in the demo.");
            Console.WriteLine("For instructions on how to set up a landing zone,");
            Console.WriteLine("see https://docs.aws.amazon.com/controltower/latest/
userguide/getting-started-from-console.html");

            // List available landing zones
            var landingZones = await wrapper!.ListLandingZonesAsync();
            if (landingZones.Count > 0)
            {
                Console.WriteLine("\nAvailable Landing Zones:");
                for (int i = 0; i < landingZones.Count; i++)
                {
                    Console.WriteLine($"{i + 1}. {landingZones[i].Arn}");
                }

                Console.Write($"\nDo you want to use the first landing zone in the
 list ({landingZones[0].Arn})? (y/n): ");
                if (GetUserConfirmation())
                {
                    useLandingZone = true;
                    Console.WriteLine($"Using landing zone: {landingZones[0].Arn}");
                    ouArn = await SetupOrganizationAsync();
                }
            }

            // Managing Baselines
            Console.WriteLine("\nManaging Baselines:");
            var baselines = await wrapper.ListBaselinesAsync();
            Console.WriteLine("\nListing available Baselines:");
            BaselineSummary? controlTowerBaseline = null;
            foreach (var baseline in baselines)
            {
                if (baseline.Name == "AWSControlTowerBaseline")
```

```
                    controlTowerBaseline = baseline;
                Console.WriteLine($"  - {baseline.Name}");
            }

            EnabledBaselineSummary? identityCenterBaseline = null;
            string? baselineArn = null;

            if (useLandingZone && ouArn != null)
            {
                Console.WriteLine("\nListing enabled baselines:");
                var enabledBaselines = await wrapper.ListEnabledBaselinesAsync();
                foreach (var baseline in enabledBaselines)
                {
                    if (baseline.BaselineIdentifier.Contains("baseline/
LN25R72TTG6IGPTQ"))
                        identityCenterBaseline = baseline;
                    Console.WriteLine($"  - {baseline.BaselineIdentifier}");
                }

                if (controlTowerBaseline != null)
                {
                    Console.Write("\nDo you want to enable the Control Tower
 Baseline? (y/n): ");
                    if (GetUserConfirmation())
                    {
                        Console.WriteLine("\nEnabling Control Tower Baseline.");
                        var icBaselineArn = identityCenterBaseline?.Arn;
                        baselineArn = await wrapper.EnableBaselineAsync(ouArn,
                            controlTowerBaseline.Arn, "5.0", icBaselineArn ?? "");
                        var alreadyEnabled = false;
                        if (baselineArn != null)
                        {
                            Console.WriteLine($"Enabled baseline ARN:
 {baselineArn}");
                        }
                        else
                        {
                            // Find the enabled baseline
                            foreach (var enabled in enabledBaselines)
                            {
                                if (enabled.BaselineIdentifier ==
 controlTowerBaseline.Arn)
                                {
                                    baselineArn = enabled.Arn;
```

```
                                    alreadyEnabled = true;
                                    Console.WriteLine("No change, the selected
baseline was already enabled.");
                                    break;
                                }
                            }
                        }

                        if (baselineArn != null)
                        {
                            Console.Write("\nDo you want to reset the Control Tower
Baseline? (y/n): ");
                            if (GetUserConfirmation())
                            {
                                Console.WriteLine($"\nResetting Control Tower
Baseline: {baselineArn}");
                                var operationId = await
wrapper.ResetEnabledBaselineAsync(baselineArn);
                                Console.WriteLine($"Reset baseline operation id:
{operationId}");
                            }

                            Console.Write("\nDo you want to disable the Control
Tower Baseline? (y/n): ");
                            if (GetUserConfirmation())
                            {
                                Console.WriteLine($"Disabling baseline ARN:
{baselineArn}");
                                var operationId = await
wrapper.DisableBaselineAsync(baselineArn);
                                Console.WriteLine($"Disabled baseline operation id:
{operationId}");
                                if (alreadyEnabled)
                                {
                                    Console.WriteLine($"\nRe-enabling Control Tower
Baseline: {baselineArn}");
                                    // Re-enable the Control Tower baseline if it
was originally enabled.
                                    await wrapper.EnableBaselineAsync(ouArn,
                                        controlTowerBaseline.Arn, "5.0",
icBaselineArn ?? "");
                                }
                            }
                        }
```

```
                    }
                }
            }

            // Managing Controls
            Console.WriteLine("\nManaging Controls:");
            var controls = await wrapper.ListControlsAsync();
            Console.WriteLine("\nListing first 5 available Controls:");
            for (int i = 0; i < Math.Min(5, controls.Count); i++)
            {
                Console.WriteLine($"{i + 1}. {controls[i].Name} -
 {controls[i].Arn}");
            }

            if (useLandingZone && ouArn != null)
            {
                var enabledControls = await wrapper.ListEnabledControlsAsync(ouArn);
                Console.WriteLine("\nListing enabled controls:");
                for (int i = 0; i < enabledControls.Count; i++)
                {
                    Console.WriteLine($"{i + 1}.
 {enabledControls[i].ControlIdentifier}");
                }

                // Find first non-enabled control
                var enabledControlArns = enabledControls.Select(c =>
 c.Arn).ToHashSet();
                var controlArn = controls.FirstOrDefault(c => !
enabledControlArns.Contains(c.Arn))?.Arn;

                if (controlArn != null)
                {
                    Console.Write($"\nDo you want to enable the control
 {controlArn}? (y/n): ");
                    if (GetUserConfirmation())
                    {
                        Console.WriteLine($"\nEnabling control: {controlArn}");
                        var operationId = await
 wrapper.EnableControlAsync(controlArn, ouArn);
                        if (operationId != null)
                        {
                            Console.WriteLine($"Enabled control with operation id:
 {operationId}");
```

```
                            Console.Write("\nDo you want to disable the control? (y/
n): ");
                            if (GetUserConfirmation())
                            {
                                Console.WriteLine("\nDisabling the control...");
                                var disableOpId = await
 wrapper.DisableControlAsync(controlArn, ouArn);
                                Console.WriteLine($"Disable operation ID:
 {disableOpId}");
                            }
                        }
                    }
                }
            }

            Console.WriteLine("\nThis concludes the example scenario.");
            Console.WriteLine("Thanks for watching!");
            Console.WriteLine(new string('-', 88));
        }
        catch (Exception ex)
        {
            logger.LogError(ex, "An error occurred during the Control Tower
 scenario.");
            Console.WriteLine($"An error occurred: {ex.Message}");
        }
    }

    /// <summary>
    /// Sets up AWS Organizations and creates or finds a Sandbox OU.
    /// </summary>
    /// <returns>The ARN of the Sandbox organizational unit.</returns>
    private static async Task<string> SetupOrganizationAsync()
    {
        Console.WriteLine("\nChecking organization status...");

        try
        {
            var orgResponse = await orgClient!.DescribeOrganizationAsync(new
 DescribeOrganizationRequest());
            var orgId = orgResponse.Organization.Id;
            Console.WriteLine($"Account is part of organization: {orgId}");
        }
        catch (AWSOrganizationsNotInUseException)
        {
```

```
            Console.WriteLine("No organization found. Creating a new
organization...");
            var createResponse = await orgClient!.CreateOrganizationAsync(new
CreateOrganizationRequest { FeatureSet = OrganizationFeatureSet.ALL });
            var orgId = createResponse.Organization.Id;
            Console.WriteLine($"Created new organization: {orgId}");
        }

        // Look for Sandbox OU
        var roots = await orgClient.ListRootsAsync(new ListRootsRequest());
        var rootId = roots.Roots[0].Id;

        Console.WriteLine("Checking for Sandbox OU...");
        var ous = await orgClient.ListOrganizationalUnitsForParentAsync(new
ListOrganizationalUnitsForParentRequest { ParentId = rootId });
        var sandboxOu = ous.OrganizationalUnits.FirstOrDefault(ou => ou.Name ==
"Sandbox");

        if (sandboxOu == null)
        {
            Console.WriteLine("Creating Sandbox OU...");
            var createOuResponse = await orgClient.CreateOrganizationalUnitAsync(new
CreateOrganizationalUnitRequest { ParentId = rootId, Name = "Sandbox" });
            sandboxOu = createOuResponse.OrganizationalUnit;
            Console.WriteLine($"Created new Sandbox OU: {sandboxOu.Id}");
        }
        else
        {
            Console.WriteLine($"Found existing Sandbox OU: {sandboxOu.Id}");
        }

        return sandboxOu.Arn;
    }

    /// <summary>
    /// Gets user confirmation by waiting for input or returning true if not
interactive.
    /// </summary>
    /// <returns>True if user enters 'y' or if isInteractive is false, otherwise
false.</returns>
    private static bool GetUserConfirmation()
    {
        return Console.ReadLine()?.ToLower() == "y" || !isInteractive;
    }
```

```
}
```

Metode pembungkus yang dipanggil oleh skenario untuk mengelola tindakan Aurora.

```
using Amazon.ControlCatalog;
using Amazon.ControlCatalog.Model;
using Amazon.ControlTower;
using Amazon.ControlTower.Model;
using ValidationException = Amazon.ControlTower.Model.ValidationException;

namespace ControlTowerActions;

/// <summary>
/// Methods to perform AWS Control Tower actions.
/// </summary>
public class ControlTowerWrapper
{
    private readonly IAmazonControlTower _controlTowerService;
    private readonly IAmazonControlCatalog _controlCatalogService;

    /// <summary>
    /// Constructor for the wrapper class containing AWS Control Tower actions.
    /// </summary>
    /// <param name="controlTowerService">The AWS Control Tower client object.</
param>
    /// <param name="controlCatalogService">The AWS Control Catalog client object.</
param>
    public ControlTowerWrapper(IAmazonControlTower controlTowerService,
 IAmazonControlCatalog controlCatalogService)
    {
        _controlTowerService = controlTowerService;
        _controlCatalogService = controlCatalogService;
    }

    /// <summary>
    /// List the AWS Control Tower landing zones for an account.
    /// </summary>
    /// <returns>A list of LandingZoneSummary objects.</returns>
    public async Task<List<LandingZoneSummary>> ListLandingZonesAsync()
    {
```

```
        try
        {
            var landingZones = new List<LandingZoneSummary>();

            var landingZonesPaginator =
_controlTowerService.Paginators.ListLandingZones(new ListLandingZonesRequest());

            await foreach (var response in landingZonesPaginator.Responses)
            {
                landingZones.AddRange(response.LandingZones);
            }

            return landingZones;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list landing zones. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// List all baselines.
    /// </summary>
    /// <returns>A list of baseline summaries.</returns>
    public async Task<List<BaselineSummary>> ListBaselinesAsync()
    {
        try
        {
            var baselines = new List<BaselineSummary>();

            var baselinesPaginator =
_controlTowerService.Paginators.ListBaselines(new ListBaselinesRequest());

            await foreach (var response in baselinesPaginator.Responses)
            {
                baselines.AddRange(response.Baselines);
            }

            return baselines;
        }
        catch (AmazonControlTowerException ex)
```

```
        {
            Console.WriteLine($"Couldn't list baselines. Here's why: {ex.ErrorCode}:
 {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// List all enabled baselines.
    /// </summary>
    /// <returns>A list of enabled baseline summaries.</returns>
    public async Task<List<EnabledBaselineSummary>> ListEnabledBaselinesAsync()
    {
        try
        {
            var enabledBaselines = new List<EnabledBaselineSummary>();

            var enabledBaselinesPaginator =
 _controlTowerService.Paginators.ListEnabledBaselines(new
 ListEnabledBaselinesRequest());

            await foreach (var response in enabledBaselinesPaginator.Responses)
            {
                enabledBaselines.AddRange(response.EnabledBaselines);
            }

            return enabledBaselines;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list enabled baselines. Here's why:
 {ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Enable a baseline for the specified target.
    /// </summary>
    /// <param name="targetIdentifier">The ARN of the target.</param>
    /// <param name="baselineIdentifier">The identifier of baseline to enable.</
param>
```

```
   /// <param name="baselineVersion">The version of baseline to enable.</param>
   /// <param name="identityCenterBaseline">The identifier of identity center
baseline if it is enabled.</param>
   /// <returns>The enabled baseline ARN or null.</returns>
   public async Task<string?> EnableBaselineAsync(string targetIdentifier, string
baselineIdentifier, string baselineVersion, string identityCenterBaseline)
   {
       try
       {
           var parameters = new List<EnabledBaselineParameter>();
           if (!string.IsNullOrEmpty(identityCenterBaseline))
           {
               parameters.Add(
                   new EnabledBaselineParameter
                   {
                       Key = "IdentityCenterEnabledBaselineArn",
                       Value = identityCenterBaseline
                   });
           }
           var request = new EnableBaselineRequest
           {
               BaselineIdentifier = baselineIdentifier,
               BaselineVersion = baselineVersion,
               TargetIdentifier = targetIdentifier,
               Parameters = parameters
           };

           var response = await _controlTowerService.EnableBaselineAsync(request);
           var operationId = response.OperationIdentifier;

           // Wait for operation to complete
           while (true)
           {
               var status = await GetBaselineOperationAsync(operationId);
               Console.WriteLine($"Baseline operation status: {status}");
               if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
               {
                   break;
               }
               await Task.Delay(30000); // Wait 30 seconds
           }

           return response.Arn;
```

```
        }
        catch (ValidationException ex)
        {
            if (ex.Message.Contains("already enabled"))
                Console.WriteLine("Baseline is already enabled for this target");
            else { Console.WriteLine(ex.Message); }
            // Write the message and return null if baseline cannot be enabled.
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't enable baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Disable a baseline for a specific target and wait for the operation to
complete.
    /// </summary>
    /// <param name="enabledBaselineIdentifier">The identifier of the baseline to
disable.</param>
    /// <returns>The operation ID or null if there was a conflict.</returns>
    public async Task<string?> DisableBaselineAsync(string
enabledBaselineIdentifier)
    {
        try
        {
            var request = new DisableBaselineRequest
            {
                EnabledBaselineIdentifier = enabledBaselineIdentifier
            };

            var response = await _controlTowerService.DisableBaselineAsync(request);
            var operationId = response.OperationIdentifier;

            // Wait for operation to complete
            while (true)
            {
                var status = await GetBaselineOperationAsync(operationId);
                Console.WriteLine($"Baseline operation status: {status}");
```

```
                if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return operationId;
        }
        catch (ConflictException ex)
        {
            Console.WriteLine($"Conflict disabling baseline: {ex.Message}. Skipping
disable step.");
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't disable baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Reset an enabled baseline for a specific target.
    /// </summary>
    /// <param name="enabledBaselineIdentifier">The identifier of the enabled
baseline to reset.</param>
    /// <returns>The operation ID.</returns>
    public async Task<string> ResetEnabledBaselineAsync(string
enabledBaselineIdentifier)
    {
        try
        {
            var request = new ResetEnabledBaselineRequest
            {
                EnabledBaselineIdentifier = enabledBaselineIdentifier
            };

            var response = await
_controlTowerService.ResetEnabledBaselineAsync(request);
            var operationId = response.OperationIdentifier;
```

```
            // Wait for operation to complete
            while (true)
            {
                var status = await GetBaselineOperationAsync(operationId);
                Console.WriteLine($"Baseline operation status: {status}");
                if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return operationId;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException)
        {
            Console.WriteLine("Target not found, unable to reset enabled
baseline.");
            throw;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't reset enabled baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Get the status of a baseline operation.
    /// </summary>
    /// <param name="operationId">The ID of the baseline operation.</param>
    /// <returns>The operation status.</returns>
    public async Task<BaselineOperationStatus> GetBaselineOperationAsync(string
operationId)
    {
        try
        {
            var request = new GetBaselineOperationRequest
            {
                OperationIdentifier = operationId
```

```
            };

            var response = await
 _controlTowerService.GetBaselineOperationAsync(request);
            return response.BaselineOperation.Status;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException)
        {
            Console.WriteLine("Operation not found.");
            throw;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't get baseline operation status. Here's why:
 {ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// List enabled controls for a target organizational unit.
    /// </summary>
    /// <param name="targetIdentifier">The target organizational unit identifier.</
param>
    /// <returns>A list of enabled control summaries.</returns>
    public async Task<List<EnabledControlSummary>> ListEnabledControlsAsync(string
 targetIdentifier)
    {
        try
        {
            var request = new ListEnabledControlsRequest
            {
                TargetIdentifier = targetIdentifier
            };

            var enabledControls = new List<EnabledControlSummary>();

            var enabledControlsPaginator =
 _controlTowerService.Paginators.ListEnabledControls(request);

            await foreach (var response in enabledControlsPaginator.Responses)
            {
                enabledControls.AddRange(response.EnabledControls);
```

```
            }

            return enabledControls;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("not registered with AWS Control Tower"))
        {
            Console.WriteLine("AWS Control Tower must be enabled to work with
enabling controls.");
            return new List<EnabledControlSummary>();
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list enabled controls. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Enable a control for a specified target.
    /// </summary>
    /// <param name="controlArn">The ARN of the control to enable.</param>
    /// <param name="targetIdentifier">The identifier of the target (e.g., OU
ARN).</param>
    /// <returns>The operation ID or null if already enabled.</returns>
    public async Task<string?> EnableControlAsync(string controlArn, string
targetIdentifier)
    {
        try
        {
            Console.WriteLine(controlArn);
            Console.WriteLine(targetIdentifier);

            var request = new EnableControlRequest
            {
                ControlIdentifier = controlArn,
                TargetIdentifier = targetIdentifier
            };

            var response = await _controlTowerService.EnableControlAsync(request);
            var operationId = response.OperationIdentifier;
```

```
            // Wait for operation to complete
            while (true)
            {
                var status = await GetControlOperationAsync(operationId);
                Console.WriteLine($"Control operation status: {status}");
                if (status == ControlOperationStatus.SUCCEEDED || status ==
ControlOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return operationId;
        }
        catch (Amazon.ControlTower.Model.ValidationException ex) when
(ex.Message.Contains("already enabled"))
        {
            Console.WriteLine("Control is already enabled for this target");
            return null;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("not registered with AWS Control Tower"))
        {
            Console.WriteLine("AWS Control Tower must be enabled to work with
enabling controls.");
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't enable control. Here's why: {ex.ErrorCode}:
{ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Disable a control for a specified target.
    /// </summary>
    /// <param name="controlArn">The ARN of the control to disable.</param>
    /// <param name="targetIdentifier">The identifier of the target (e.g., OU
ARN).</param>
    /// <returns>The operation ID.</returns>
```

```
   public async Task<string> DisableControlAsync(string controlArn, string
targetIdentifier)
   {
       try
       {
           var request = new DisableControlRequest
           {
               ControlIdentifier = controlArn,
               TargetIdentifier = targetIdentifier
           };

           var response = await _controlTowerService.DisableControlAsync(request);
           var operationId = response.OperationIdentifier;

           // Wait for operation to complete
           while (true)
           {
               var status = await GetControlOperationAsync(operationId);
               Console.WriteLine($"Control operation status: {status}");
               if (status == ControlOperationStatus.SUCCEEDED || status ==
ControlOperationStatus.FAILED)
               {
                   break;
               }
               await Task.Delay(30000); // Wait 30 seconds
           }

           return operationId;
       }
       catch (Amazon.ControlTower.Model.ResourceNotFoundException)
       {
           Console.WriteLine("Control not found.");
           throw;
       }
       catch (AmazonControlTowerException ex)
       {
           Console.WriteLine($"Couldn't disable control. Here's why:
{ex.ErrorCode}: {ex.Message}");
           throw;
       }
   }


   /// <summary>
```

```
    /// Get the status of a control operation.
    /// </summary>
    /// <param name="operationId">The ID of the control operation.</param>
    /// <returns>The operation status.</returns>
    public async Task<ControlOperationStatus> GetControlOperationAsync(string
operationId)
    {
        try
        {
            var request = new GetControlOperationRequest
            {
                OperationIdentifier = operationId
            };

            var response = await
_controlTowerService.GetControlOperationAsync(request);
            return response.ControlOperation.Status;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException)
        {
            Console.WriteLine("Operation not found.");
            throw;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't get control operation status. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// List all controls in the Control Tower control catalog.
    /// </summary>
    /// <returns>A list of control summaries.</returns>
    public async Task<List<ControlSummary>> ListControlsAsync()
    {
        try
        {
            var controls = new List<ControlSummary>();
```

```
            var controlsPaginator =
_controlCatalogService.Paginators.ListControls(new
Amazon.ControlCatalog.Model.ListControlsRequest());

            await foreach (var response in controlsPaginator.Responses)
            {
                controls.AddRange(response.Controls);
            }

            return controls;
        }
        catch (AmazonControlCatalogException ex)
        {
            Console.WriteLine($"Couldn't list controls. Here's why: {ex.ErrorCode}:
{ex.Message}");
            throw;
        }
    }

}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk .NET .
  - CreateLandingZone
  - DeleteLandingZone
  - DisableBaseline
  - DisableControl
  - EnableBaseline
  - EnableControl
  - GetControlOperation
  - GetLandingZoneOperation
  - ListBaselines
  - ListEnabledBaselines
  - ListEnabledControls
  - ListLandingZones
  - ResetEnabledBaseline

Tindakan

`DisableBaseline`

Contoh kode berikut menunjukkan cara menggunakan`DisableBaseline`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   /// <summary>
   /// Disable a baseline for a specific target and wait for the operation to
complete.
   /// </summary>
   /// <param name="enabledBaselineIdentifier">The identifier of the baseline to
disable.</param>
   /// <returns>The operation ID or null if there was a conflict.</returns>
   public async Task<string?> DisableBaselineAsync(string
enabledBaselineIdentifier)
   {
       try
       {
           var request = new DisableBaselineRequest
           {
               EnabledBaselineIdentifier = enabledBaselineIdentifier
           };

           var response = await _controlTowerService.DisableBaselineAsync(request);
           var operationId = response.OperationIdentifier;

           // Wait for operation to complete
           while (true)
           {
               var status = await GetBaselineOperationAsync(operationId);
               Console.WriteLine($"Baseline operation status: {status}");
               if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
               {
```

```
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return operationId;
        }
        catch (ConflictException ex)
        {
            Console.WriteLine($"Conflict disabling baseline: {ex.Message}. Skipping
disable step.");
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't disable baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- Untuk detail API, lihat DisableBaselinedi Referensi AWS SDK untuk .NET API.

`DisableControl`

Contoh kode berikut menunjukkan cara menggunakan`DisableControl`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Disable a control for a specified target.
    /// </summary>
    /// <param name="controlArn">The ARN of the control to disable.</param>
```

```
   /// <param name="targetIdentifier">The identifier of the target (e.g., OU
ARN).</param>
   /// <returns>The operation ID.</returns>
   public async Task<string> DisableControlAsync(string controlArn, string
targetIdentifier)
   {
       try
       {
           var request = new DisableControlRequest
           {
               ControlIdentifier = controlArn,
               TargetIdentifier = targetIdentifier
           };

           var response = await _controlTowerService.DisableControlAsync(request);
           var operationId = response.OperationIdentifier;

           // Wait for operation to complete
           while (true)
           {
               var status = await GetControlOperationAsync(operationId);
               Console.WriteLine($"Control operation status: {status}");
               if (status == ControlOperationStatus.SUCCEEDED || status ==
ControlOperationStatus.FAILED)
               {
                   break;
               }
               await Task.Delay(30000); // Wait 30 seconds
           }

           return operationId;
       }
       catch (Amazon.ControlTower.Model.ResourceNotFoundException)
       {
           Console.WriteLine("Control not found.");
           throw;
       }
       catch (AmazonControlTowerException ex)
       {
           Console.WriteLine($"Couldn't disable control. Here's why:
{ex.ErrorCode}: {ex.Message}");
           throw;
       }
   }
```

- Untuk detail API, lihat DisableControldi Referensi AWS SDK untuk .NET API.

`EnableBaseline`

Contoh kode berikut menunjukkan cara menggunakan`EnableBaseline`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Enable a baseline for the specified target.
    /// </summary>
    /// <param name="targetIdentifier">The ARN of the target.</param>
    /// <param name="baselineIdentifier">The identifier of baseline to enable.</
param>
    /// <param name="baselineVersion">The version of baseline to enable.</param>
    /// <param name="identityCenterBaseline">The identifier of identity center
 baseline if it is enabled.</param>
    /// <returns>The enabled baseline ARN or null.</returns>
    public async Task<string?> EnableBaselineAsync(string targetIdentifier, string
 baselineIdentifier, string baselineVersion, string identityCenterBaseline)
    {
        try
        {
            var parameters = new List<EnabledBaselineParameter>();
            if (!string.IsNullOrEmpty(identityCenterBaseline))
            {
                parameters.Add(
                    new EnabledBaselineParameter
                    {
                        Key = "IdentityCenterEnabledBaselineArn",
                        Value = identityCenterBaseline
                    });
            }
```

```
            var request = new EnableBaselineRequest
            {
                BaselineIdentifier = baselineIdentifier,
                BaselineVersion = baselineVersion,
                TargetIdentifier = targetIdentifier,
                Parameters = parameters
            };

            var response = await _controlTowerService.EnableBaselineAsync(request);
            var operationId = response.OperationIdentifier;

            // Wait for operation to complete
            while (true)
            {
                var status = await GetBaselineOperationAsync(operationId);
                Console.WriteLine($"Baseline operation status: {status}");
                if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return response.Arn;
        }
        catch (ValidationException ex)
        {
            if (ex.Message.Contains("already enabled"))
                Console.WriteLine("Baseline is already enabled for this target");
            else { Console.WriteLine(ex.Message); }
            // Write the message and return null if baseline cannot be enabled.
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't enable baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- Untuk detail API, lihat EnableBaselinedi Referensi AWS SDK untuk .NET API.

`EnableControl`

Contoh kode berikut menunjukkan cara menggunakan`EnableControl`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   /// <summary>
   /// Enable a control for a specified target.
   /// </summary>
   /// <param name="controlArn">The ARN of the control to enable.</param>
   /// <param name="targetIdentifier">The identifier of the target (e.g., OU
ARN).</param>
   /// <returns>The operation ID or null if already enabled.</returns>
   public async Task<string?> EnableControlAsync(string controlArn, string
targetIdentifier)
   {
       try
       {
           Console.WriteLine(controlArn);
           Console.WriteLine(targetIdentifier);

           var request = new EnableControlRequest
           {
               ControlIdentifier = controlArn,
               TargetIdentifier = targetIdentifier
           };

           var response = await _controlTowerService.EnableControlAsync(request);
           var operationId = response.OperationIdentifier;

           // Wait for operation to complete
           while (true)
           {
               var status = await GetControlOperationAsync(operationId);
```

```
                Console.WriteLine($"Control operation status: {status}");
                if (status == ControlOperationStatus.SUCCEEDED || status ==
ControlOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return operationId;
        }
        catch (Amazon.ControlTower.Model.ValidationException ex) when
(ex.Message.Contains("already enabled"))
        {
            Console.WriteLine("Control is already enabled for this target");
            return null;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("not registered with AWS Control Tower"))
        {
            Console.WriteLine("AWS Control Tower must be enabled to work with
enabling controls.");
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't enable control. Here's why: {ex.ErrorCode}:
{ex.Message}");
            throw;
        }
    }
```

- Untuk detail API, lihat EnableControldi Referensi AWS SDK untuk .NET API.

`GetBaselineOperation`

Contoh kode berikut menunjukkan cara menggunakan`GetBaselineOperation`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   /// <summary>
   /// Get the status of a baseline operation.
   /// </summary>
   /// <param name="operationId">The ID of the baseline operation.</param>
   /// <returns>The operation status.</returns>
   public async Task<BaselineOperationStatus> GetBaselineOperationAsync(string
operationId)
   {
       try
       {
           var request = new GetBaselineOperationRequest
           {
               OperationIdentifier = operationId
           };

           var response = await
_controlTowerService.GetBaselineOperationAsync(request);
           return response.BaselineOperation.Status;
       }
       catch (Amazon.ControlTower.Model.ResourceNotFoundException)
       {
           Console.WriteLine("Operation not found.");
           throw;
       }
       catch (AmazonControlTowerException ex)
       {
           Console.WriteLine($"Couldn't get baseline operation status. Here's why:
{ex.ErrorCode}: {ex.Message}");
           throw;
       }
   }
```

- Untuk detail API, lihat GetBaselineOperationdi Referensi AWS SDK untuk .NET API.

`GetControlOperation`

Contoh kode berikut menunjukkan cara menggunakan`GetControlOperation`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   /// <summary>
   /// Get the status of a control operation.
   /// </summary>
   /// <param name="operationId">The ID of the control operation.</param>
   /// <returns>The operation status.</returns>
   public async Task<ControlOperationStatus> GetControlOperationAsync(string
operationId)
   {
       try
       {
           var request = new GetControlOperationRequest
           {
               OperationIdentifier = operationId
           };

           var response = await
_controlTowerService.GetControlOperationAsync(request);
           return response.ControlOperation.Status;
       }
       catch (Amazon.ControlTower.Model.ResourceNotFoundException)
       {
           Console.WriteLine("Operation not found.");
           throw;
       }
       catch (AmazonControlTowerException ex)
       {
           Console.WriteLine($"Couldn't get control operation status. Here's why:
{ex.ErrorCode}: {ex.Message}");
```

```
            throw;
        }
    }
```

- Untuk detail API, lihat GetControlOperationdi Referensi AWS SDK untuk .NET API.

`ListBaselines`

Contoh kode berikut menunjukkan cara menggunakan`ListBaselines`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// List all baselines.
    /// </summary>
    /// <returns>A list of baseline summaries.</returns>
    public async Task<List<BaselineSummary>> ListBaselinesAsync()
    {
        try
        {
            var baselines = new List<BaselineSummary>();

            var baselinesPaginator =
_controlTowerService.Paginators.ListBaselines(new ListBaselinesRequest());

            await foreach (var response in baselinesPaginator.Responses)
            {
                baselines.AddRange(response.Baselines);
            }

            return baselines;
        }
        catch (AmazonControlTowerException ex)
        {
```

```
            Console.WriteLine($"Couldn't list baselines. Here's why: {ex.ErrorCode}:
{ex.Message}");
            throw;
        }
    }
```

- Untuk detail API, lihat ListBaselinesdi Referensi AWS SDK untuk .NET API.

`ListEnabledBaselines`

Contoh kode berikut menunjukkan cara menggunakan`ListEnabledBaselines`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// List all enabled baselines.
    /// </summary>
    /// <returns>A list of enabled baseline summaries.</returns>
    public async Task<List<EnabledBaselineSummary>> ListEnabledBaselinesAsync()
    {
        try
        {
            var enabledBaselines = new List<EnabledBaselineSummary>();

            var enabledBaselinesPaginator =
_controlTowerService.Paginators.ListEnabledBaselines(new
ListEnabledBaselinesRequest());

            await foreach (var response in enabledBaselinesPaginator.Responses)
            {
                enabledBaselines.AddRange(response.EnabledBaselines);
            }

            return enabledBaselines;
```

```
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list enabled baselines. Here's why:
 {ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- Untuk detail API, lihat ListEnabledBaselinesdi Referensi AWS SDK untuk .NET API.

`ListEnabledControls`

Contoh kode berikut menunjukkan cara menggunakan`ListEnabledControls`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// List enabled controls for a target organizational unit.
    /// </summary>
    /// <param name="targetIdentifier">The target organizational unit identifier.</
param>
    /// <returns>A list of enabled control summaries.</returns>
    public async Task<List<EnabledControlSummary>> ListEnabledControlsAsync(string
 targetIdentifier)
    {
        try
        {
            var request = new ListEnabledControlsRequest
            {
                TargetIdentifier = targetIdentifier
            };
```

```
            var enabledControls = new List<EnabledControlSummary>();

            var enabledControlsPaginator =
_controlTowerService.Paginators.ListEnabledControls(request);

            await foreach (var response in enabledControlsPaginator.Responses)
            {
                enabledControls.AddRange(response.EnabledControls);
            }

            return enabledControls;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("not registered with AWS Control Tower"))
        {
            Console.WriteLine("AWS Control Tower must be enabled to work with
enabling controls.");
            return new List<EnabledControlSummary>();
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list enabled controls. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- Untuk detail API, lihat ListEnabledControlsdi Referensi AWS SDK untuk .NET API.

`ListLandingZones`

Contoh kode berikut menunjukkan cara menggunakan`ListLandingZones`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// List the AWS Control Tower landing zones for an account.
    /// </summary>
    /// <returns>A list of LandingZoneSummary objects.</returns>
    public async Task<List<LandingZoneSummary>> ListLandingZonesAsync()
    {
        try
        {
            var landingZones = new List<LandingZoneSummary>();

            var landingZonesPaginator =
_controlTowerService.Paginators.ListLandingZones(new ListLandingZonesRequest());

            await foreach (var response in landingZonesPaginator.Responses)
            {
                landingZones.AddRange(response.LandingZones);
            }

            return landingZones;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list landing zones. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- Untuk detail API, lihat ListLandingZonesdi Referensi AWS SDK untuk .NET API.

`ResetEnabledBaseline`

Contoh kode berikut menunjukkan cara menggunakan`ResetEnabledBaseline`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   /// <summary>
   /// Reset an enabled baseline for a specific target.
   /// </summary>
   /// <param name="enabledBaselineIdentifier">The identifier of the enabled
baseline to reset.</param>
   /// <returns>The operation ID.</returns>
   public async Task<string> ResetEnabledBaselineAsync(string
enabledBaselineIdentifier)
   {
       try
       {
           var request = new ResetEnabledBaselineRequest
           {
               EnabledBaselineIdentifier = enabledBaselineIdentifier
           };

           var response = await
_controlTowerService.ResetEnabledBaselineAsync(request);
           var operationId = response.OperationIdentifier;

           // Wait for operation to complete
           while (true)
           {
               var status = await GetBaselineOperationAsync(operationId);
               Console.WriteLine($"Baseline operation status: {status}");
               if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
               {
                   break;
               }
               await Task.Delay(30000); // Wait 30 seconds
           }

           return operationId;
```

```
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException)
        {
            Console.WriteLine("Target not found, unable to reset enabled
baseline.");
            throw;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't reset enabled baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- Untuk detail API, lihat ResetEnabledBaselinedi Referensi AWS SDK untuk .NET API.

Contoh SDK untuk .NET DynamoDB menggunakan (v4)

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK untuk .NET (v4) dengan DynamoDB.

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

Tindakan merupakan 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

- Memulai
- Hal-hal mendasar
- Tindakan

Memulai

Halo DynamoDB

Contoh kode berikut menunjukkan bagaimana untuk memulai menggunakan DynamoDB.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Microsoft.Extensions.DependencyInjection;

namespace DynamoDBActions;

/// <summary>
/// A simple example that demonstrates basic DynamoDB operations.
/// </summary>
public class HelloDynamoDB
{
    /// <summary>
    /// HelloDynamoDB lists the existing DynamoDB tables for the default user.
    /// </summary>
    /// <param name="args">Command line arguments</param>
    /// <returns>Async task.</returns>
    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon DynamoDB.
        using var host =
 Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonDynamoDB>()
            )
            .Build();

        // Now the client is available for injection.
```

```
        var dynamoDbClient = host.Services.GetRequiredService<IAmazonDynamoDB>();

        try
        {
            var request = new ListTablesRequest();
            var tableNames = new List<string>();

            var paginatorForTables = dynamoDbClient.Paginators.ListTables(request);

            await foreach (var tableName in paginatorForTables.TableNames)
            {
                tableNames.Add(tableName);
            }

            Console.WriteLine("Welcome to the DynamoDB Hello Service example. " +
                              "\nLet's list your DynamoDB tables:");
            tableNames.ForEach(table =>
            {
                Console.WriteLine($"Table: {table}");
            });
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB service error occurred while
 listing tables. {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while listing tables.
 {ex.Message}");
        }
    }
}
```

- Untuk detail API, lihat ListTablesdi Referensi AWS SDK untuk .NET API.

Hal-hal mendasar

Pelajari dasar-dasarnya

Contoh kode berikut ini menunjukkan cara untuk melakukan:

- Buat tabel yang dapat menyimpan data film.
- Masukkan, dapatkan, dan perbarui satu film dalam tabel tersebut.
- Tulis data film ke tabel dari file JSON sampel.
- Kueri untuk film yang dirilis pada tahun tertentu.
- Pindai film yang dirilis dalam suatu rentang tahun.
- Hapus film dari tabel, lalu hapus tabel tersebut.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
/// <summary>
/// This example application performs the following basic Amazon DynamoDB
/// functions:
///     CreateTableAsync
///     PutItemAsync
///     UpdateItemAsync
///     BatchWriteItemAsync
///     GetItemAsync
///     DeleteItemAsync
///     Query
///     Scan
///     DeleteItemAsync.
/// </summary>
public class DynamoDbBasics
{
    public static bool IsInteractive = true;

    // Separator for the console display.
    private static readonly string SepBar = new string('-', 80);

    /// <summary>
    /// The main entry point for the DynamoDB Basics example application.
    /// </summary>
```

```
    /// <param name="args">Command line arguments.</param>
    /// <returns>A task representing the asynchronous operation.</returns>
    public static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon DynamoDB.
        using var host =
Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonDynamoDB>()
                    .AddTransient<DynamoDbWrapper>())
            .Build();

        // Now the wrapper is available for injection.
        var dynamoDbWrapper = host.Services.GetRequiredService<DynamoDbWrapper>();

        var tableName = "movie_table";

        var movieFileName = @"movies.json";

        DisplayInstructions();

        // Create a new table and wait for it to be active.
        Console.WriteLine($"Creating the new table: {tableName}");

        var success = await dynamoDbWrapper.CreateMovieTableAsync(tableName);

        Console.WriteLine(success
            ? $"\nTable: {tableName} successfully created."
            : $"\nCould not create {tableName}.");

        WaitForEnter();

        // Add a single new movie to the table.
        var newMovie = new Movie
        {
            Year = 2021,
            Title = "Spider-Man: No Way Home",
        };

        success = await dynamoDbWrapper.PutItemAsync(newMovie, tableName);
        if (success)
        {
            Console.WriteLine($"Added {newMovie.Title} to the table.");
        }
```

```
        else
        {
            Console.WriteLine("Could not add movie to table.");
        }

        WaitForEnter();

        // Update the new movie by adding a plot and rank.
        var newInfo = new MovieInfo
        {
            Plot = "With Spider-Man's identity now revealed, Peter asks" +
                   "Doctor Strange for help. When a spell goes wrong, dangerous" +
                   "foes from other worlds start to appear, forcing Peter to" +
                   "discover what it truly means to be Spider-Man.",
            Rank = 9,
        };

        success = await dynamoDbWrapper.UpdateItemAsync(newMovie, newInfo,
tableName);
        if (success)
        {
            Console.WriteLine($"Successfully updated the movie: {newMovie.Title}");
        }
        else
        {
            Console.WriteLine("Could not update the movie.");
        }

        WaitForEnter();

        // Add a batch of movies to the DynamoDB table from a list of
        // movies in a JSON file.
        var itemCount = await dynamoDbWrapper.BatchWriteItemsAsync(movieFileName,
tableName);
        Console.WriteLine($"Added {itemCount} movies to the table.");

        WaitForEnter();

        // Get a movie by key. (partition + sort)
        var lookupMovie = new Movie
        {
            Title = "Jurassic Park",
            Year = 1993,
        };
```

```
        Console.WriteLine("Looking for the movie \"Jurassic Park\".");
        var item = await dynamoDbWrapper.GetItemAsync(lookupMovie, tableName);
        if (item?.Count > 0)
        {
            dynamoDbWrapper.DisplayItem(item);
        }
        else
        {
            Console.WriteLine($"Couldn't find {lookupMovie.Title}");
        }

        WaitForEnter();

        // Delete a movie.
        var movieToDelete = new Movie
        {
            Title = "The Town",
            Year = 2010,
        };

        success = await dynamoDbWrapper.DeleteItemAsync(tableName, movieToDelete);

        if (success)
        {
            Console.WriteLine($"Successfully deleted {movieToDelete.Title}.");
        }
        else
        {
            Console.WriteLine($"Could not delete {movieToDelete.Title}.");
        }

        WaitForEnter();

        // Use Query to find all the movies released in 2010.
        int findYear = 2010;
        Console.WriteLine($"Movies released in {findYear}");
        var queryCount = await dynamoDbWrapper.QueryMoviesAsync(tableName,
findYear);
        Console.WriteLine($"Found {queryCount} movies released in {findYear}");

        WaitForEnter();

        // Use Scan to get a list of movies from 2001 to 2011.
```

```
        int startYear = 2001;
        int endYear = 2011;
        var scanCount = await dynamoDbWrapper.ScanTableAsync(tableName, startYear,
endYear);
        Console.WriteLine($"Found {scanCount} movies released between {startYear}
and {endYear}");

        WaitForEnter();

        // Delete the table.
        success = await dynamoDbWrapper.DeleteTableAsync(tableName);

        if (success)
        {
            Console.WriteLine($"Successfully deleted {tableName}");
        }
        else
        {
            Console.WriteLine($"Could not delete {tableName}");
        }

        Console.WriteLine("The DynamoDB Basics example application is complete.");

        WaitForEnter();
    }

    /// <summary>
    /// Displays the description of the application on the console.
    /// </summary>
    private static void DisplayInstructions()
    {
        if (!IsInteractive)
        {
            return;
        }

        Console.Clear();
        Console.WriteLine();
        Console.Write(new string(' ', 28));
        Console.WriteLine("DynamoDB Basics Example");
        Console.WriteLine(SepBar);
        Console.WriteLine("This demo application shows the basics of using DynamoDB
with the AWS SDK.");
        Console.WriteLine(SepBar);
```

```
        Console.WriteLine("The application does the following:");
        Console.WriteLine("\t1. Creates a table with partition: year and
 sort:title.");
        Console.WriteLine("\t2. Adds a single movie to the table.");
        Console.WriteLine("\t3. Adds movies to the table from moviedata.json.");
        Console.WriteLine("\t4. Updates the rating and plot of the movie that was
 just added.");
        Console.WriteLine("\t5. Gets a movie using its key (partition + sort).");
        Console.WriteLine("\t6. Deletes a movie.");
        Console.WriteLine("\t7. Uses QueryAsync to return all movies released in a
 given year.");
        Console.WriteLine("\t8. Uses ScanAsync to return all movies released within
 a range of years.");
        Console.WriteLine("\t9. Finally, it deletes the table that was just
 created.");
        WaitForEnter();
    }

    /// <summary>
    /// Simple method to wait for the Enter key to be pressed.
    /// </summary>
    private static void WaitForEnter()
    {
        if (IsInteractive)
        {
            Console.WriteLine("\nPress <Enter> to continue.");
            Console.WriteLine(SepBar);
            _ = Console.ReadLine();
        }
    }
}
```

Gunakan klien yang disuntikkan untuk operasi tabel.

```
using System.Text.Json;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.DynamoDBv2.Model;

namespace DynamoDBActions;
```

```
/// <summary>
/// Methods of this class perform Amazon DynamoDB operations.
/// </summary>
public class DynamoDbWrapper
{
    private readonly IAmazonDynamoDB _amazonDynamoDB;

    /// <summary>
    /// Constructor for the DynamoDbWrapper class.
    /// </summary>
    /// <param name="amazonDynamoDB">The injected DynamoDB client.</param>
    public DynamoDbWrapper(IAmazonDynamoDB amazonDynamoDB)
    {
        _amazonDynamoDB = amazonDynamoDB;
    }
```

Membuat tabel yang akan berisi data film.

```
    /// <summary>
    /// Creates a new Amazon DynamoDB table and then waits for the new
    /// table to become active.
    /// </summary>
    /// <param name="tableName">The name of the table to create.</param>
    /// <returns>A Boolean value indicating the success of the operation.</returns>
    public async Task<bool> CreateMovieTableAsync(string tableName)
    {
        try
        {
            var response = await _amazonDynamoDB.CreateTableAsync(new
 CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                {
                    new AttributeDefinition
                    {
                        AttributeName = "title",
                        AttributeType = ScalarAttributeType.S,
                    },
                    new AttributeDefinition
```

```
                    {
                        AttributeName = "year",
                        AttributeType = ScalarAttributeType.N,
                    },
                },
                KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement
                    {
                        AttributeName = "year",
                        KeyType = KeyType.HASH,
                    },
                    new KeySchemaElement
                    {
                        AttributeName = "title",
                        KeyType = KeyType.RANGE,
                    },
                },
                BillingMode = BillingMode.PAY_PER_REQUEST,
            });

            // Wait until the table is ACTIVE and then report success.
            Console.Write("Waiting for table to become active...");

            var request = new DescribeTableRequest
            {
                TableName = response.TableDescription.TableName,
            };

            TableStatus status;

            int sleepDuration = 2000;

            do
            {
                Thread.Sleep(sleepDuration);

                var describeTableResponse = await
_amazonDynamoDB.DescribeTableAsync(request);
                status = describeTableResponse.Table.TableStatus;

                Console.Write(".");
            }
            while (status != "ACTIVE");
```

```
            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException ex)
        {
            Console.WriteLine($"Table {tableName} already exists. {ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while creating
 table {tableName}. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating table {tableName}.
 {ex.Message}");
            throw;
        }
    }
```

Menambahkan satu film ke tabel.

```
    /// <summary>
    /// Adds a new item to the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing informtation for
    /// the movie to add to the table.</param>
    /// <param name="tableName">The name of the table where the item will be
 added.</param>
    /// <returns>A Boolean value that indicates the results of adding the item.</
returns>
    public async Task<bool> PutItemAsync(Movie newMovie, string tableName)
    {
        try
        {
            var item = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
```

```
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
            };

            await _amazonDynamoDB.PutItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while putting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while putting item.
{ex.Message}");
            throw;
        }
    }
```

Memperbarui satu item dalam tabel.

```
    /// <summary>
    /// Updates an existing item in the movies table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information for
    /// the movie to update.</param>
    /// <param name="newInfo">A MovieInfo object that contains the
    /// information that will be changed.</param>
```

```
    /// <param name="tableName">The name of the table that contains the movie.</
param>
    /// <returns>A Boolean value that indicates the success of the operation.</
returns>
    public async Task<bool> UpdateItemAsync(
        Movie newMovie,
        MovieInfo newInfo,
        string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };
            var updates = new Dictionary<string, AttributeValueUpdate>
            {
                ["info.plot"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { S = newInfo.Plot },
                },

                ["info.rating"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { N = newInfo.Rank.ToString() },
                },
            };

            var request = new UpdateItemRequest
            {
                AttributeUpdates = updates,
                Key = key,
                TableName = tableName,
            };

            await _amazonDynamoDB.UpdateItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
```

```
            Console.WriteLine($"Table {tableName} or item was not found.
{ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while updating
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while updating item.
{ex.Message}");
            throw;
        }
    }
```

Mengambil satu item dari tabel film.

```
    /// <summary>
    /// Gets information about an existing movie from the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information about
    /// the movie to retrieve.</param>
    /// <param name="tableName">The name of the table containing the movie.</param>
    /// <returns>A Dictionary object containing information about the item
    /// retrieved.</returns>
    public async Task<Dictionary<string, AttributeValue>> GetItemAsync(Movie
newMovie, string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new GetItemRequest
```

```
            {
                Key = key,
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.GetItemAsync(request);
            return response.Item;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return new Dictionary<string, AttributeValue>();
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while getting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while getting item.
{ex.Message}");
            throw;
        }
    }
```

Menulis batch item ke tabel film.

```
    /// <summary>
    /// Loads the contents of a JSON file into a list of movies to be
    /// added to the DynamoDB table.
    /// </summary>
    /// <param name="movieFileName">The name of the JSON file.</param>
    /// <returns>A generic list of movie objects.</returns>
    public List<Movie> ImportMovies(string movieFileName)
    {
        var moviesList = new List<Movie>();
        if (!File.Exists(movieFileName))
        {
```

```
            return moviesList;
        }

        using var sr = new StreamReader(movieFileName);
        string json = sr.ReadToEnd();
        var allMovies = JsonSerializer.Deserialize<List<Movie>>(
            json,
            new JsonSerializerOptions
            {
                PropertyNameCaseInsensitive = true
            });

        // Now return the first 250 entries.
        if (allMovies != null && allMovies.Any())
        {
            moviesList = allMovies.GetRange(0, 250);
        }
        return moviesList;
    }

    /// <summary>
    /// Writes 250 items to the movie table.
    /// </summary>
    /// <param name="movieFileName">A string containing the full path to
    /// the JSON file containing movie data.</param>
    /// <param name="tableName">The name of the table to write items to.</param>
    /// <returns>A long integer value representing the number of movies
    /// imported from the JSON file.</returns>
    public async Task<long> BatchWriteItemsAsync(
        string movieFileName, string tableName)
    {
        try
        {
            var movies = ImportMovies(movieFileName);
            if (!movies.Any())
            {
                Console.WriteLine("Couldn't find the JSON file with movie data.");
                return 0;
            }

            var context = new DynamoDBContextBuilder()
                // Optional call to provide a specific instance of IAmazonDynamoDB
                .WithDynamoDBClient(() => _amazonDynamoDB)
                .Build();
```

```
            var movieBatch = context.CreateBatchWrite<Movie>(
                new BatchWriteConfig()
                {
                    OverrideTableName = tableName
                });
            movieBatch.AddPutItems(movies);

            Console.WriteLine("Adding imported movies to the table.");
            await movieBatch.ExecuteAsync();

            return movies.Count;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table was not found during batch write operation.
{ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred during batch write
operation. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred during batch write operation.
{ex.Message}");
            throw;
        }
    }
```

Menghapus satu item dari tabel.

```
    /// <summary>
    /// Deletes a single item from a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table from which the item
    /// will be deleted.</param>
```

```
   /// <param name="movieToDelete">A movie object containing the title and
   /// year of the movie to delete.</param>
   /// <returns>A Boolean value indicating the success or failure of the
   /// delete operation.</returns>
   public async Task<bool> DeleteItemAsync(
       string tableName,
       Movie movieToDelete)
   {
       try
       {
           var key = new Dictionary<string, AttributeValue>
           {
               ["title"] = new AttributeValue { S = movieToDelete.Title },
               ["year"] = new AttributeValue { N = movieToDelete.Year.ToString() },
           };

           var request = new DeleteItemRequest { TableName = tableName, Key =
key, };

           await _amazonDynamoDB.DeleteItemAsync(request);
           return true;
       }
       catch (ResourceNotFoundException ex)
       {
           Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
           return false;
       }
       catch (AmazonDynamoDBException ex)
       {
           Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
item. {ex.Message}");
           throw;
       }
       catch (Exception ex)
       {
           Console.WriteLine($"An error occurred while deleting item.
{ex.Message}");
           throw;
       }
   }
```

Melakukan kueri tabel untuk film yang dirilis pada tahun tertentu.

```
/// <summary>
/// Queries the table for movies released in a particular year and
/// then displays the information for the movies returned.
/// </summary>
/// <param name="tableName">The name of the table to query.</param>
/// <param name="year">The release year for which we want to
/// view movies.</param>
/// <returns>The number of movies that match the query.</returns>
public async Task<int> QueryMoviesAsync(string tableName, int year)
{
    try
    {
        var movieTable = new TableBuilder(_amazonDynamoDB, tableName)
            .AddHashKey("year", DynamoDBEntryType.Numeric)
            .AddRangeKey("title", DynamoDBEntryType.String)
            .Build();

        var filter = new QueryFilter("year", QueryOperator.Equal, year);

        Console.WriteLine("\nFind movies released in: {year}:");

        var config = new QueryOperationConfig()
        {
            Limit = 10, // 10 items per page.
            Select = SelectValues.SpecificAttributes,
            AttributesToGet = new List<string>
            {
                "title",
                "year",
            },
            ConsistentRead = true,
            Filter = filter,
        };

        // Value used to track how many movies match the
        // supplied criteria.
        var moviesFound = 0;

        var search = movieTable.Query(config);
        do
        {
```

```
                var movieList = await search.GetNextSetAsync();
                moviesFound += movieList.Count;

                foreach (var movie in movieList)
                {
                    DisplayDocument(movie);
                }
            }
            while (!search.IsDone);

            return moviesFound;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while querying
movies. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while querying movies.
{ex.Message}");
            throw;
        }
    }
```

Memindai tabel untuk film yang dirilis dalam suatu rentang tahun.

```
    /// <summary>
    /// Scans the table for movies released between the specified years.
    /// </summary>
    /// <param name="tableName">The name of the table to scan.</param>
    /// <param name="startYear">The starting year for the range.</param>
    /// <param name="endYear">The ending year for the range.</param>
    /// <returns>The number of movies found in the specified year range.</returns>
    public async Task<int> ScanTableAsync(
```

```
        string tableName,
        int startYear,
        int endYear)
    {
        try
        {
            var request = new ScanRequest
            {
                TableName = tableName,
                ExpressionAttributeNames = new Dictionary<string, string>
                {
                    { "#yr", "year" },
                },
                ExpressionAttributeValues = new Dictionary<string, AttributeValue>
                {
                    { ":y_a", new AttributeValue { N = startYear.ToString() } },
                    { ":y_z", new AttributeValue { N = endYear.ToString() } },
                },
                FilterExpression = "#yr between :y_a and :y_z",
                ProjectionExpression = "#yr, title, info.actors[0], info.directors,
info.running_time_secs",
                Limit = 10 // Set a limit to demonstrate using the LastEvaluatedKey.
            };

            // Keep track of how many movies were found.
            int foundCount = 0;

            var response = new ScanResponse();
            do
            {
                response = await _amazonDynamoDB.ScanAsync(request);
                foundCount += response.Items.Count;
                response.Items.ForEach(i => DisplayItem(i));
                request.ExclusiveStartKey = response.LastEvaluatedKey;
            }
            while (response?.LastEvaluatedKey?.Count > 0);
            return foundCount;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
```

```
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while scanning
table. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while scanning table.
{ex.Message}");
            throw;
        }
    }
```

Menghapus tabel film.

```
    /// <summary>
    /// Deletes a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table to delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</returns>
    public async Task<bool> DeleteTableAsync(string tableName)
    {
        try
        {
            var request = new DeleteTableRequest
            {
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.DeleteTableAsync(request);

            Console.WriteLine($"Table {response.TableDescription.TableName}
successfully deleted.");
            return true;

        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found and cannot be
deleted. {ex.Message}");
            return false;
```

```
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
table {tableName}. {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting table {tableName}.
{ex.Message}");
            return false;
        }
    }
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk .NET .
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Kueri
  - Scan
  - UpdateItem

Tindakan

`BatchWriteItem`

Contoh kode berikut menunjukkan cara menggunakan`BatchWriteItem`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Menulis batch item ke tabel film.

```
/// <summary>
/// Loads the contents of a JSON file into a list of movies to be
/// added to the DynamoDB table.
/// </summary>
/// <param name="movieFileName">The name of the JSON file.</param>
/// <returns>A generic list of movie objects.</returns>
public List<Movie> ImportMovies(string movieFileName)
{
    var moviesList = new List<Movie>();
    if (!File.Exists(movieFileName))
    {
        return moviesList;
    }

    using var sr = new StreamReader(movieFileName);
    string json = sr.ReadToEnd();
    var allMovies = JsonSerializer.Deserialize<List<Movie>>(
        json,
        new JsonSerializerOptions
        {
            PropertyNameCaseInsensitive = true
        });

    // Now return the first 250 entries.
    if (allMovies != null && allMovies.Any())
    {
        moviesList = allMovies.GetRange(0, 250);
    }
    return moviesList;
}

/// <summary>
```

```
    /// Writes 250 items to the movie table.
    /// </summary>
    /// <param name="movieFileName">A string containing the full path to
    /// the JSON file containing movie data.</param>
    /// <param name="tableName">The name of the table to write items to.</param>
    /// <returns>A long integer value representing the number of movies
    /// imported from the JSON file.</returns>
    public async Task<long> BatchWriteItemsAsync(
        string movieFileName, string tableName)
    {
        try
        {
            var movies = ImportMovies(movieFileName);
            if (!movies.Any())
            {
                Console.WriteLine("Couldn't find the JSON file with movie data.");
                return 0;
            }

            var context = new DynamoDBContextBuilder()
                // Optional call to provide a specific instance of IAmazonDynamoDB
                .WithDynamoDBClient(() => _amazonDynamoDB)
                .Build();

            var movieBatch = context.CreateBatchWrite<Movie>(
                new BatchWriteConfig()
                {
                    OverrideTableName = tableName
                });
            movieBatch.AddPutItems(movies);

            Console.WriteLine("Adding imported movies to the table.");
            await movieBatch.ExecuteAsync();

            return movies.Count;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table was not found during batch write operation.
{ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
```

```
            Console.WriteLine($"An Amazon DynamoDB error occurred during batch write
operation. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred during batch write operation.
{ex.Message}");
            throw;
        }
    }
```

- Untuk detail API, lihat BatchWriteItemdi Referensi AWS SDK untuk .NET API.

`CreateTable`

Contoh kode berikut menunjukkan cara menggunakan`CreateTable`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Creates a new Amazon DynamoDB table and then waits for the new
    /// table to become active.
    /// </summary>
    /// <param name="tableName">The name of the table to create.</param>
    /// <returns>A Boolean value indicating the success of the operation.</returns>
    public async Task<bool> CreateMovieTableAsync(string tableName)
    {
        try
        {
            var response = await _amazonDynamoDB.CreateTableAsync(new
CreateTableRequest
            {
```

```
            TableName = tableName,
            AttributeDefinitions = new List<AttributeDefinition>()
            {
                new AttributeDefinition
                {
                    AttributeName = "title",
                    AttributeType = ScalarAttributeType.S,
                },
                new AttributeDefinition
                {
                    AttributeName = "year",
                    AttributeType = ScalarAttributeType.N,
                },
            },
            KeySchema = new List<KeySchemaElement>()
            {
                new KeySchemaElement
                {
                    AttributeName = "year",
                    KeyType = KeyType.HASH,
                },
                new KeySchemaElement
                {
                    AttributeName = "title",
                    KeyType = KeyType.RANGE,
                },
            },
            BillingMode = BillingMode.PAY_PER_REQUEST,
        });

        // Wait until the table is ACTIVE and then report success.
        Console.Write("Waiting for table to become active...");

        var request = new DescribeTableRequest
        {
            TableName = response.TableDescription.TableName,
        };

        TableStatus status;

        int sleepDuration = 2000;

        do
        {
```

```
                Thread.Sleep(sleepDuration);

                var describeTableResponse = await
_amazonDynamoDB.DescribeTableAsync(request);
                status = describeTableResponse.Table.TableStatus;

                Console.Write(".");
            }
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException ex)
        {
            Console.WriteLine($"Table {tableName} already exists. {ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while creating
table {tableName}. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating table {tableName}.
{ex.Message}");
            throw;
        }
    }
```

- Untuk detail API, lihat CreateTabledi Referensi AWS SDK untuk .NET API.

`DeleteItem`

Contoh kode berikut menunjukkan cara menggunakan`DeleteItem`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Deletes a single item from a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table from which the item
    /// will be deleted.</param>
    /// <param name="movieToDelete">A movie object containing the title and
    /// year of the movie to delete.</param>
    /// <returns>A Boolean value indicating the success or failure of the
    /// delete operation.</returns>
    public async Task<bool> DeleteItemAsync(
        string tableName,
        Movie movieToDelete)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = movieToDelete.Title },
                ["year"] = new AttributeValue { N = movieToDelete.Year.ToString() },
            };

            var request = new DeleteItemRequest { TableName = tableName, Key =
key, };

            await _amazonDynamoDB.DeleteItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
```

```
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting item.
{ex.Message}");
            throw;
        }
    }
```

- Untuk detail API, lihat DeleteItemdi Referensi AWS SDK untuk .NET API.

`DeleteTable`

Contoh kode berikut menunjukkan cara menggunakan`DeleteTable`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Deletes a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table to delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</returns>
    public async Task<bool> DeleteTableAsync(string tableName)
    {
        try
        {
            var request = new DeleteTableRequest
            {
                TableName = tableName,
            };
```

```
            var response = await _amazonDynamoDB.DeleteTableAsync(request);

            Console.WriteLine($"Table {response.TableDescription.TableName}
successfully deleted.");
            return true;

        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found and cannot be
deleted. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
table {tableName}. {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting table {tableName}.
{ex.Message}");
            return false;
        }
    }
```

- Untuk detail API, lihat DeleteTabledi Referensi AWS SDK untuk .NET API.

`GetItem`

Contoh kode berikut menunjukkan cara menggunakan`GetItem`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   /// <summary>
   /// Gets information about an existing movie from the table.
   /// </summary>
   /// <param name="newMovie">A Movie object containing information about
   /// the movie to retrieve.</param>
   /// <param name="tableName">The name of the table containing the movie.</param>
   /// <returns>A Dictionary object containing information about the item
   /// retrieved.</returns>
   public async Task<Dictionary<string, AttributeValue>> GetItemAsync(Movie
newMovie, string tableName)
   {
       try
       {
           var key = new Dictionary<string, AttributeValue>
           {
               ["title"] = new AttributeValue { S = newMovie.Title },
               ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
           };

           var request = new GetItemRequest
           {
               Key = key,
               TableName = tableName,
           };

           var response = await _amazonDynamoDB.GetItemAsync(request);
           return response.Item;
       }
       catch (ResourceNotFoundException ex)
       {
           Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
           return new Dictionary<string, AttributeValue>();
       }
       catch (AmazonDynamoDBException ex)
       {
           Console.WriteLine($"An Amazon DynamoDB error occurred while getting
item. {ex.Message}");
           throw;
       }
       catch (Exception ex)
       {
```

```
            Console.WriteLine($"An error occurred while getting item.
 {ex.Message}");
            throw;
        }
    }
```

- Untuk detail API, lihat GetItemdi Referensi AWS SDK untuk .NET API.

`PutItem`

Contoh kode berikut menunjukkan cara menggunakan`PutItem`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Adds a new item to the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing informtation for
    /// the movie to add to the table.</param>
    /// <param name="tableName">The name of the table where the item will be
 added.</param>
    /// <returns>A Boolean value that indicates the results of adding the item.</
returns>
    public async Task<bool> PutItemAsync(Movie newMovie, string tableName)
    {
        try
        {
            var item = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };
```

```
            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
            };

            await _amazonDynamoDB.PutItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while putting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while putting item.
{ex.Message}");
            throw;
        }
    }
```

- Untuk detail API, lihat PutItemdi Referensi AWS SDK untuk .NET API.

`Query`

Contoh kode berikut menunjukkan cara menggunakan`Query`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
/// <summary>
/// Queries the table for movies released in a particular year and
/// then displays the information for the movies returned.
/// </summary>
/// <param name="tableName">The name of the table to query.</param>
/// <param name="year">The release year for which we want to
/// view movies.</param>
/// <returns>The number of movies that match the query.</returns>
public async Task<int> QueryMoviesAsync(string tableName, int year)
{
    try
    {
        var movieTable = new TableBuilder(_amazonDynamoDB, tableName)
            .AddHashKey("year", DynamoDBEntryType.Numeric)
            .AddRangeKey("title", DynamoDBEntryType.String)
            .Build();

        var filter = new QueryFilter("year", QueryOperator.Equal, year);

        Console.WriteLine("\nFind movies released in: {year}:");

        var config = new QueryOperationConfig()
        {
            Limit = 10, // 10 items per page.
            Select = SelectValues.SpecificAttributes,
            AttributesToGet = new List<string>
            {
                "title",
                "year",
            },
            ConsistentRead = true,
            Filter = filter,
        };

        // Value used to track how many movies match the
        // supplied criteria.
        var moviesFound = 0;

        var search = movieTable.Query(config);
        do
        {
            var movieList = await search.GetNextSetAsync();
```

```
                moviesFound += movieList.Count;

                foreach (var movie in movieList)
                {
                    DisplayDocument(movie);
                }
            }
            while (!search.IsDone);

            return moviesFound;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while querying
movies. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while querying movies.
{ex.Message}");
            throw;
        }
    }
```

- Untuk detail API, lihat Kueri di Referensi API AWS SDK untuk .NET .

Scan

Contoh kode berikut menunjukkan cara menggunakan`Scan`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Scans the table for movies released between the specified years.
    /// </summary>
    /// <param name="tableName">The name of the table to scan.</param>
    /// <param name="startYear">The starting year for the range.</param>
    /// <param name="endYear">The ending year for the range.</param>
    /// <returns>The number of movies found in the specified year range.</returns>
    public async Task<int> ScanTableAsync(
        string tableName,
        int startYear,
        int endYear)
    {
        try
        {
            var request = new ScanRequest
            {
                TableName = tableName,
                ExpressionAttributeNames = new Dictionary<string, string>
                {
                    { "#yr", "year" },
                },
                ExpressionAttributeValues = new Dictionary<string, AttributeValue>
                {
                    { ":y_a", new AttributeValue { N = startYear.ToString() } },
                    { ":y_z", new AttributeValue { N = endYear.ToString() } },
                },
                FilterExpression = "#yr between :y_a and :y_z",
                ProjectionExpression = "#yr, title, info.actors[0], info.directors,
info.running_time_secs",
                Limit = 10 // Set a limit to demonstrate using the LastEvaluatedKey.
            };

            // Keep track of how many movies were found.
            int foundCount = 0;
```

```
            var response = new ScanResponse();
            do
            {
                response = await _amazonDynamoDB.ScanAsync(request);
                foundCount += response.Items.Count;
                response.Items.ForEach(i => DisplayItem(i));
                request.ExclusiveStartKey = response.LastEvaluatedKey;
            }
            while (response?.LastEvaluatedKey?.Count > 0);
            return foundCount;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while scanning
table. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while scanning table.
{ex.Message}");
            throw;
        }
    }
```

- Untuk detail API, lihat Scan di Referensi API AWS SDK untuk .NET .

`UpdateItem`

Contoh kode berikut menunjukkan cara menggunakan`UpdateItem`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Updates an existing item in the movies table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information for
    /// the movie to update.</param>
    /// <param name="newInfo">A MovieInfo object that contains the
    /// information that will be changed.</param>
    /// <param name="tableName">The name of the table that contains the movie.</
param>
    /// <returns>A Boolean value that indicates the success of the operation.</
returns>
    public async Task<bool> UpdateItemAsync(
        Movie newMovie,
        MovieInfo newInfo,
        string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };
            var updates = new Dictionary<string, AttributeValueUpdate>
            {
                ["info.plot"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { S = newInfo.Plot },
                },

                ["info.rating"] = new AttributeValueUpdate
                {
```

```
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { N = newInfo.Rank.ToString() },
                },
            };

            var request = new UpdateItemRequest
            {
                AttributeUpdates = updates,
                Key = key,
                TableName = tableName,
            };

            await _amazonDynamoDB.UpdateItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} or item was not found.
{ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while updating
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while updating item.
{ex.Message}");
            throw;
        }
    }
```

- Untuk detail API, lihat UpdateItemdi Referensi AWS SDK untuk .NET API.

Contoh Amazon EC2 menggunakan SDK untuk .NET (v4)

Contoh kode berikut menunjukkan cara melakukan tindakan dan menerapkan skenario umum dengan menggunakan AWS SDK untuk .NET (v4) dengan Amazon EC2.

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

Topik

- Memulai

Memulai

Halo Amazon EC2

Contoh kode berikut menunjukkan cara memulai menggunakan Amazon EC2.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
namespace EC2Actions;

public class HelloEc2
{
    /// <summary>
    /// HelloEc2 lists the existing security groups for the default users.
    /// </summary>
    /// <param name="args">Command line arguments</param>
    /// <returns>Async task.</returns>
    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon Elastic Compute Cloud (Amazon
 EC2).
```

```
        using var host =
Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonEC2>()
                .AddTransient<EC2Wrapper>()
            )
            .Build();

        // Now the client is available for injection.
        var ec2Client = host.Services.GetRequiredService<IAmazonEC2>();

        try
        {
            // Retrieve information for up to 10 Amazon EC2 security groups.
            var request = new DescribeSecurityGroupsRequest { MaxResults = 10 };
            var securityGroups = new List<SecurityGroup>();

            var paginatorForSecurityGroups =
                ec2Client.Paginators.DescribeSecurityGroups(request);

            await foreach (var securityGroup in
paginatorForSecurityGroups.SecurityGroups)
            {
                securityGroups.Add(securityGroup);
            }

            // Now print the security groups returned by the call to
            // DescribeSecurityGroupsAsync.
            Console.WriteLine("Welcome to the EC2 Hello Service example. " +
                              "\nLet's list your Security Groups:");
            securityGroups.ForEach(group =>
            {
                Console.WriteLine(
                    $"Security group: {group.GroupName} ID: {group.GroupId}");
            });
        }
        catch (AmazonEC2Exception ex)
        {
            Console.WriteLine($"An Amazon EC2 service error occurred while listing
security groups. {ex.Message}");
        }
        catch (Exception ex)
        {
```

```
            Console.WriteLine($"An error occurred while listing security groups.
 {ex.Message}");
        }
    }
}
```

- Untuk detail API, lihat DescribeSecurityGroupsdi Referensi AWS SDK untuk .NET API.

Contoh Amazon ECS menggunakan SDK untuk .NET (v4)

Contoh kode berikut menunjukkan cara melakukan tindakan dan menerapkan skenario umum dengan menggunakan AWS SDK untuk .NET (v4) dengan Amazon ECS.

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

Topik

- Memulai

Memulai

Halo Amazon ECS

Contoh kode berikut menunjukkan cara memulai menggunakan Amazon ECS.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
using Amazon.ECS;
using Amazon.ECS.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
```

```
using Microsoft.Extensions.Logging.Debug;

namespace ECSActions;

/// <summary>
/// A class that introduces the Amazon ECS Client by listing the
/// cluster ARNs for the account.
/// </summary>
public class HelloECS
{
    static async System.Threading.Tasks.Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
 the Amazon ECS client.
        // Use your AWS profile name, or leave it blank to use the default profile.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonECS>()
            )
            .Build();

        var amazonECSClient = host.Services.GetRequiredService<IAmazonECS>();

        Console.WriteLine($"Hello Amazon ECS! Following are some cluster ARNS
 available in the your account");
        Console.WriteLine();

        var clusters = new List<string>();

        var clustersPaginator = amazonECSClient.Paginators.ListClusters(new
 ListClustersRequest());

        await foreach (var response in clustersPaginator.Responses)
        {
            clusters.AddRange(response.ClusterArns);
        }

        if (clusters.Count > 0)
        {
```

```
            clusters.ForEach(cluster =>
            {
                Console.WriteLine($"\tARN: {cluster}");
                Console.WriteLine($"Cluster Name: {cluster.Split("/").Last()}");
                Console.WriteLine();
            });
        }
        else
        {
            Console.WriteLine("No clusters were found.");
        }

    }
}
```

- Untuk detail API, lihat ListClustersdi Referensi AWS SDK untuk .NET API.

AWS IoT contoh menggunakan SDK untuk .NET (v4)

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK untuk .NET (v4) dengan AWS IoT.

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

Tindakan merupakan 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

- Memulai
- Hal-hal mendasar
- Tindakan

Memulai

Halo AWS IoT

Contoh kode berikut menunjukkan bagaimana untuk mulai menggunakan AWS IoT.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
/// <summary>
/// Hello AWS IoT example.
/// </summary>
public class HelloIoT
{
    /// <summary>
    /// Main method to run the Hello IoT example.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A Task object.</returns>
    public static async Task Main(string[] args)
    {
        var iotClient = new AmazonIoTClient();

        try
        {
            Console.WriteLine("Hello AWS IoT! Let's list your IoT Things:");
            Console.WriteLine(new string('-', 80));

            // Use pages of 10.
            var request = new ListThingsRequest()
            {
                MaxResults = 10
            };
            var response = await iotClient.ListThingsAsync(request);

            // Since there is not a built-in paginator, use the NextMarker to
 paginate.
```

```
            bool hasMoreResults = true;

            var things = new List<ThingAttribute>();
            while (hasMoreResults)
            {
                things.AddRange(response.Things);

                // If NextMarker is not null, there are more results. Get the next
page of results.
                if (!String.IsNullOrEmpty(response.NextMarker))
                {
                    request.Marker = response.NextMarker;
                    response = await iotClient.ListThingsAsync(request);
                }
                else
                    hasMoreResults = false;
            }

            if (things is { Count: > 0 })
            {
                Console.WriteLine($"Found {things.Count} IoT Things:");
                foreach (var thing in things)
                {
                    Console.WriteLine($"- Thing Name: {thing.ThingName}");
                    Console.WriteLine($"  Thing ARN: {thing.ThingArn}");
                    Console.WriteLine($"  Thing Type: {thing.ThingTypeName ?? "No
type specified"}");
                    Console.WriteLine($"  Version: {thing.Version}");

                    if (thing.Attributes?.Count > 0)
                    {
                        Console.WriteLine("  Attributes:");
                        foreach (var attr in thing.Attributes)
                        {
                            Console.WriteLine($"    {attr.Key}: {attr.Value}");
                        }
                    }
                    Console.WriteLine();
                }
            }
            else
            {
                Console.WriteLine("No IoT Things found in your account.");
```

```
                Console.WriteLine("You can create IoT Things using the IoT Basics
 scenario example.");
            }

            Console.WriteLine("Hello IoT completed successfully.");
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            Console.WriteLine($"Request throttled, please try again later:
 {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't list Things. Here's why: {ex.Message}");
        }
    }
}
```

- Untuk detail API, lihat ListThings di Referensi AWS SDK untuk .NET API.

Hal-hal mendasar

Pelajari dasar-dasarnya

Contoh kode berikut ini menunjukkan cara untuk melakukan:

- Buat AWS IoT Sesuatu.
- Hasilkan sertifikat perangkat.
- Perbarui AWS IoT Sesuatu dengan Atribut.
- Kembalikan titik akhir yang unik.
- Buat daftar AWS IoT sertifikat Anda.
- Perbarui AWS IoT bayangan.
- Tuliskan informasi negara.
- Membuat aturan.
- Buat daftar aturan Anda.
- Cari sesuatu menggunakan nama Thing.
- Hapus AWS IoT sesuatu.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Jalankan skenario interaktif yang menunjukkan AWS IoT fitur.

```
/// <summary>
/// Scenario class for AWS IoT basics.
/// </summary>
public class IoTBasics
{
    public static bool IsInteractive = true;
    public static IoTWrapper? Wrapper = null;
    public static IAmazonCloudFormation? CloudFormationClient = null;
    public static ILogger<IoTBasics> logger = null!;
    private static IoTWrapper _iotWrapper = null!;
    private static IAmazonCloudFormation _amazonCloudFormation = null!;
    private static ILogger<IoTBasics> _logger = null!;

    private static string _stackName = "IoTBasicsStack";
    private static string _stackResourcePath = "../../../../../../scenarios/basics/
iot/iot_usecase/resources/cfn_template.yaml";

    /// <summary>
    /// Main method for the IoT Basics scenario.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A Task object.</returns>
    public static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonIoT>(new AWSOptions() { Region =
 RegionEndpoint.USEast1 })
                    .AddAWSService<IAmazonCloudFormation>()
                        .AddTransient<IoTWrapper>()
                        .AddLogging(builder => builder.AddConsole())
                        .AddSingleton<IAmazonIotData>(sp =>
```

```
                        {
                            var iotService = sp.GetRequiredService<IAmazonIoT>();
                            var request = new DescribeEndpointRequest
                            {
                                EndpointType = "iot:Data-ATS"
                            };
                            var response =
 iotService.DescribeEndpointAsync(request).Result;
                            return new AmazonIotDataClient($"https://
{response.EndpointAddress}/");
                        })
            )
            .Build();

        logger = LoggerFactory.Create(builder => builder.AddConsole())
            .CreateLogger<IoTBasics>();

        Wrapper = host.Services.GetRequiredService<IoTWrapper>();
        CloudFormationClient =
 host.Services.GetRequiredService<IAmazonCloudFormation>();

        // Set the private fields for backwards compatibility
        _logger = logger;
        _iotWrapper = Wrapper;
        _amazonCloudFormation = CloudFormationClient;

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the AWS IoT example scenario.");
        Console.WriteLine("This example program demonstrates various interactions
 with the AWS Internet of Things (IoT) Core service.");
        Console.WriteLine();
        if (IsInteractive)
        {
            Console.WriteLine("Press Enter to continue...");
            Console.ReadLine();
        }
        Console.WriteLine(new string('-', 80));

        try
        {
            await RunScenarioAsync();
        }
        catch (Exception ex)
        {
```

```
            _logger.LogError(ex, "There was a problem running the scenario.");
            Console.WriteLine($"\nAn error occurred: {ex.Message}");
        }

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("The AWS IoT scenario has successfully completed.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Run the IoT Basics scenario.
    /// </summary>
    /// <returns>A Task object.</returns>
    public static async Task RunScenarioAsync()
    {
        // Use static properties if available, otherwise use private fields
        var iotWrapper = Wrapper ?? _iotWrapper;
        var cloudFormationClient = CloudFormationClient ?? _amazonCloudFormation;
        var scenarioLogger = logger ?? _logger;

        await RunScenarioInternalAsync(iotWrapper, cloudFormationClient,
 scenarioLogger);
    }

    /// <summary>
    /// Internal method to run the IoT Basics scenario with injected dependencies.
    /// </summary>
    /// <param name="iotWrapper">The IoT wrapper instance.</param>
    /// <param name="cloudFormationClient">The CloudFormation client instance.</
param>
    /// <param name="scenarioLogger">The logger instance.</param>
    /// <returns>A Task object.</returns>
    private static async Task RunScenarioInternalAsync(IoTWrapper iotWrapper,
 IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        string thingName = $"iot-thing-{Guid.NewGuid():N}";
        string certificateArn = "";
        string certificateId = "";
        string ruleName = $"iotruledefault";
        string snsTopicArn = "";

        try
        {
            // Step 1: Create an AWS IoT Thing
```

```
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("1. Create an AWS IoT Thing.");
            Console.WriteLine("An AWS IoT Thing represents a virtual entity in the
AWS IoT service that can be associated with a physical device.");
            Console.WriteLine();

            if (IsInteractive)
            {
                Console.Write("Enter Thing name: ");
                var userInput = Console.ReadLine();
                if (!string.IsNullOrEmpty(userInput))
                    thingName = userInput;
            }
            else
            {
                Console.WriteLine($"Using default Thing name: {thingName}");
            }

            var thingArn = await iotWrapper.CreateThingAsync(thingName);
            Console.WriteLine($"{thingName} was successfully created. The ARN value
is {thingArn}");
            Console.WriteLine(new string('-', 80));

            // Step 1.1: List AWS IoT Things
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("2. List AWS IoT Things.");
            Console.WriteLine("Now let's list the IoT Things to see the Thing we
just created.");
            Console.WriteLine();
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var things = await iotWrapper.ListThingsAsync();
            Console.WriteLine($"Found {things.Count} IoT Things:");
            foreach (var thing in things.Take(10)) // Show first 10 things
            {
                Console.WriteLine($"Thing Name: {thing.ThingName}");
                Console.WriteLine($"Thing ARN: {thing.ThingArn}");
                if (thing.Attributes != null && thing.Attributes.Any())
                {
                    Console.WriteLine("Attributes:");
```

```
                    foreach (var attr in thing.Attributes)
                    {
                        Console.WriteLine($"  {attr.Key}: {attr.Value}");
                    }
                }
                Console.WriteLine("--------------");
            }
            Console.WriteLine();
            Console.WriteLine(new string('-', 80));

            // Step 2: Generate a Device Certificate
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("3. Generate a device certificate.");
            Console.WriteLine("A device certificate performs a role in securing the
communication between devices (Things) and the AWS IoT platform.");
            Console.WriteLine();

            var createCert = "y";
            if (IsInteractive)
            {
                Console.Write($"Do you want to create a certificate for {thingName}?
(y/n)");
                createCert = Console.ReadLine();
            }
            else
            {
                Console.WriteLine($"Creating certificate for {thingName}...");
            }

            if (createCert?.ToLower() == "y")
            {
                var certificateResult = await
iotWrapper.CreateKeysAndCertificateAsync();
                if (certificateResult.HasValue)
                {
                    var (certArn, certPem, certId) = certificateResult.Value;
                    certificateArn = certArn;
                    certificateId = certId;

                    Console.WriteLine($"\nCertificate:");
                    // Show only first few lines of certificate for brevity
                    var lines = certPem.Split('\n');
                    for (int i = 0; i < Math.Min(lines.Length, 5); i++)
                    {
```

```
                    Console.WriteLine(lines[i]);
                }
                if (lines.Length > 5)
                {
                    Console.WriteLine("...");
                }

                Console.WriteLine($"\nCertificate ARN:");
                Console.WriteLine(certificateArn);

                // Step 3: Attach the Certificate to the AWS IoT Thing
                Console.WriteLine("Attach the certificate to the AWS IoT
Thing.");
                var attachResult = await
iotWrapper.AttachThingPrincipalAsync(thingName, certificateArn);
                if (attachResult)
                {
                    Console.WriteLine("Certificate attached to Thing
successfully.");
                }
                else
                {
                    Console.WriteLine("Failed to attach certificate to Thing.");
                }

                Console.WriteLine("Thing Details:");
                Console.WriteLine($"Thing Name: {thingName}");
                Console.WriteLine($"Thing ARN: {thingArn}");
            }
            else
            {
                Console.WriteLine("Failed to create certificate.");
            }
        }
        Console.WriteLine(new string('-', 80));

        // Step 4: Update an AWS IoT Thing with Attributes
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("4. Update an AWS IoT Thing with Attributes.");
        Console.WriteLine("IoT Thing attributes, represented as key-value pairs,
offer a pivotal advantage in facilitating efficient data");
        Console.WriteLine("management and retrieval within the AWS IoT
ecosystem.");
        Console.WriteLine();
```

```
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var attributes = new Dictionary<string, string>
            {
                { "Location", "Seattle" },
                { "DeviceType", "Sensor" },
                { "Firmware", "1.2.3" }
            };

            await iotWrapper.UpdateThingAsync(thingName, attributes);
            Console.WriteLine("Thing attributes updated successfully.");
            Console.WriteLine(new string('-', 80));

            // Step 5: Return a unique endpoint specific to the Amazon Web Services
account
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("5. Return a unique endpoint specific to the Amazon
Web Services account.");
            Console.WriteLine();
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var endpoint = await iotWrapper.DescribeEndpointAsync();
            if (endpoint != null)
            {
                var subdomain = endpoint.Split('.')[0];
                Console.WriteLine($"Extracted subdomain: {subdomain}");
                Console.WriteLine($"Full Endpoint URL: https://{endpoint}");
            }
            else
            {
                Console.WriteLine("Failed to retrieve endpoint.");
            }
            Console.WriteLine(new string('-', 80));

            // Step 6: List your AWS IoT certificates
            Console.WriteLine(new string('-', 80));
```

```
            Console.WriteLine("6. List your AWS IoT certificates");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var certificates = await iotWrapper.ListCertificatesAsync();
            foreach (var cert in certificates.Take(5)) // Show first 5 certificates
            {
                Console.WriteLine($"Cert id: {cert.CertificateId}");
                Console.WriteLine($"Cert Arn: {cert.CertificateArn}");
            }
            Console.WriteLine();
            Console.WriteLine(new string('-', 80));

            // Step 7: Create an IoT shadow
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("7. Update an IoT shadow that refers to a digital
representation or virtual twin of a physical IoT device");
            Console.WriteLine();
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var shadowPayload = JsonSerializer.Serialize(new
            {
                state = new
                {
                    desired = new
                    {
                        temperature = 25,
                        humidity = 50
                    }
                }
            });

            await iotWrapper.UpdateThingShadowAsync(thingName, shadowPayload);
            Console.WriteLine("Thing Shadow updated successfully.");
            Console.WriteLine(new string('-', 80));

            // Step 8: Write out the state information, in JSON format
```

```
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("8. Write out the state information, in JSON
format.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var shadowData = await iotWrapper.GetThingShadowAsync(thingName);
            Console.WriteLine($"Received Shadow Data: {shadowData}");
            Console.WriteLine(new string('-', 80));

            // Step 9: Set up resources (SNS topic and IAM role) and create a rule
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("9. Set up resources and create a rule");
            Console.WriteLine();

            // Deploy CloudFormation stack to create SNS topic and IAM role
            Console.WriteLine("Deploying CloudFormation stack to create SNS topic
and IAM role...");

            var deployStack = !IsInteractive || GetYesNoResponse("Would you like to
deploy the CloudFormation stack? (y/n) ");
            if (deployStack)
            {
                if (IsInteractive)
                {
                    Console.Write(
                        $"Enter stack resource file path (or press Enter for default
'{_stackResourcePath}'): ");
                    var userResourcePath = Console.ReadLine();
                    if (!string.IsNullOrEmpty(userResourcePath))
                        _stackResourcePath = userResourcePath;
                }

                _stackName = PromptUserForStackName();

                var deploySuccess = await DeployCloudFormationStack(_stackName,
cloudFormationClient, scenarioLogger);

                if (deploySuccess)
                {
                    // Get stack outputs
```

```
                    var stackOutputs = await GetStackOutputs(_stackName,
cloudFormationClient, scenarioLogger);
                    if (stackOutputs != null)
                    {
                        snsTopicArn = stackOutputs["SNSTopicArn"];
                        string roleArn = stackOutputs["RoleArn"];

                        Console.WriteLine($"Successfully deployed stack. SNS topic:
{snsTopicArn}");
                        Console.WriteLine($"Successfully deployed stack. IAM role:
{roleArn}");

                        if (IsInteractive)
                        {
                            Console.Write($"Enter Rule name (press Enter for default
'{ruleName}'): ");
                            var userRuleName = Console.ReadLine();
                            if (!string.IsNullOrEmpty(userRuleName))
                                ruleName = userRuleName;
                        }
                        else
                        {
                            Console.WriteLine($"Using default rule name:
{ruleName}");
                        }

                        // Now create the IoT rule with the CloudFormation outputs
                        var ruleResult = await
iotWrapper.CreateTopicRuleAsync(ruleName, snsTopicArn, roleArn);
                        if (ruleResult)
                        {
                            Console.WriteLine("IoT Rule created successfully.");
                        }
                        else
                        {
                            Console.WriteLine("Failed to create IoT rule.");
                        }
                    }
                    else
                    {
                        Console.WriteLine("Failed to get stack outputs. Skipping
rule creation.");
                    }
                }
```

```
                else
                {
                    Console.WriteLine("Failed to deploy CloudFormation stack.
 Skipping rule creation.");
                }
            }
            else
            {
                Console.WriteLine("Skipping CloudFormation stack deployment and rule
 creation.");
            }
            Console.WriteLine(new string('-', 80));

            // Step 10: List your rules
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("10. List your rules.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var rules = await iotWrapper.ListTopicRulesAsync();
            Console.WriteLine("List of IoT Rules:");
            foreach (var rule in rules.Take(5)) // Show first 5 rules
            {
                Console.WriteLine($"Rule Name: {rule.RuleName}");
                Console.WriteLine($"Rule ARN: {rule.RuleArn}");
                Console.WriteLine("--------------");
            }
            Console.WriteLine();
            Console.WriteLine(new string('-', 80));

            // Step 11: Search things using the Thing name
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("11. Search things using the Thing name.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var searchResults = await iotWrapper.SearchIndexAsync($"thingName:
{thingName}");
```

```
            if (searchResults.Any())
            {
                Console.WriteLine($"Thing id found using search is
{searchResults.First().ThingId}");
            }
            else
            {
                Console.WriteLine($"No search results found for Thing:
{thingName}");
            }
            Console.WriteLine(new string('-', 80));

            // Step 12: Cleanup - Detach and delete certificate
            if (!string.IsNullOrEmpty(certificateArn))
            {
                Console.WriteLine(new string('-', 80));
                var deleteCert = "y";
                if (IsInteractive)
                {
                    Console.Write($"Do you want to detach and delete the certificate
for {thingName}? (y/n)");
                    deleteCert = Console.ReadLine();
                }
                else
                {
                    Console.WriteLine($"Detaching and deleting certificate for
{thingName}...");
                }

                if (deleteCert?.ToLower() == "y")
                {
                    Console.WriteLine("12. You selected to detach and delete the
certificate.");
                    if (IsInteractive)
                    {
                        Console.WriteLine("Press Enter to continue...");
                        Console.ReadLine();
                    }

                    await iotWrapper.DetachThingPrincipalAsync(thingName,
certificateArn);
                    Console.WriteLine($"{certificateArn} was successfully removed
from {thingName}");
```

```
                    await iotWrapper.DeleteCertificateAsync(certificateId);
                    Console.WriteLine($"{certificateArn} was successfully
deleted.");
                }
                Console.WriteLine(new string('-', 80));
            }

            // Step 13: Delete the AWS IoT Thing
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("13. Delete the AWS IoT Thing.");
            var deleteThing = "y";
            if (IsInteractive)
            {
                Console.Write($"Do you want to delete the IoT Thing? (y/n)");
                deleteThing = Console.ReadLine();
            }
            else
            {
                Console.WriteLine($"Deleting IoT Thing {thingName}...");
            }

            if (deleteThing?.ToLower() == "y")
            {
                await iotWrapper.DeleteThingAsync(thingName);
                Console.WriteLine($"Deleted Thing {thingName}");
            }
            Console.WriteLine(new string('-', 80));

            // Step 14: Clean up CloudFormation stack
            if (!string.IsNullOrEmpty(snsTopicArn))
            {
                Console.WriteLine(new string('-', 80));
                Console.WriteLine("14. Clean up CloudFormation stack.");
                Console.WriteLine("Deleting the CloudFormation stack and all
resources...");

                var cleanup = !IsInteractive || GetYesNoResponse("Do you want to
delete the CloudFormation stack and all resources? (y/n) ");
                if (cleanup)
                {
                    var ruleCleanupSuccess = await
iotWrapper.DeleteTopicRuleAsync(ruleName);
```

```
                    var stackCleanupSuccess = await
 DeleteCloudFormationStack(_stackName, cloudFormationClient, scenarioLogger);
                    if (ruleCleanupSuccess && stackCleanupSuccess)
                    {
                        Console.WriteLine("Successfully cleaned up CloudFormation
 stack and all resources.");
                    }
                    else
                    {
                        Console.WriteLine("Some cleanup operations failed. Check the
 logs for details.");
                    }
                }
                else
                {
                    Console.WriteLine($"Resources will remain. Stack name:
 {_stackName}");
                }
                Console.WriteLine(new string('-', 80));
            }
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, "Error occurred during scenario
 execution.");

            // Cleanup on error
            if (!string.IsNullOrEmpty(certificateArn) && !
string.IsNullOrEmpty(thingName))
            {
                try
                {
                    await iotWrapper.DetachThingPrincipalAsync(thingName,
 certificateArn);
                    await iotWrapper.DeleteCertificateAsync(certificateId);
                }
                catch (Exception cleanupEx)
                {
                    scenarioLogger.LogError(cleanupEx, "Error during cleanup.");
                }
            }

            if (!string.IsNullOrEmpty(thingName))
            {
```

```
                try
                {
                    await iotWrapper.DeleteThingAsync(thingName);
                }
                catch (Exception cleanupEx)
                {
                    scenarioLogger.LogError(cleanupEx, "Error during Thing
cleanup.");
                }
            }

            // Clean up CloudFormation stack on error
            if (!string.IsNullOrEmpty(snsTopicArn))
            {
                try
                {
                    await _iotWrapper.DeleteTopicRuleAsync(ruleName);
                    await DeleteCloudFormationStack(_stackName,
cloudFormationClient, scenarioLogger);
                }
                catch (Exception cleanupEx)
                {
                    scenarioLogger.LogError(cleanupEx, "Error during CloudFormation
stack cleanup.");
                }
            }

            throw;
        }
    }

    /// <summary>
    /// Deploys the CloudFormation stack with the necessary resources.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>True if the stack was deployed successfully.</returns>
    private static async Task<bool> DeployCloudFormationStack(string stackName,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        Console.WriteLine($"\nDeploying CloudFormation stack: {stackName}");

        try
```

```
        {
            var request = new CreateStackRequest
            {
                StackName = stackName,
                TemplateBody = await File.ReadAllTextAsync(_stackResourcePath),
                Capabilities = new List<string> { Capability.CAPABILITY_NAMED_IAM }
            };

            var response = await cloudFormationClient.CreateStackAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"CloudFormation stack creation started:
{stackName}");

                bool stackCreated = await WaitForStackCompletion(response.StackId,
cloudFormationClient, scenarioLogger);

                if (stackCreated)
                {
                    Console.WriteLine("CloudFormation stack created successfully.");
                    return true;
                }
                else
                {
                    scenarioLogger.LogError($"CloudFormation stack creation failed:
{stackName}");
                    return false;
                }
            }
            else
            {
                scenarioLogger.LogError($"Failed to create CloudFormation stack:
{stackName}");
                return false;
            }
        }
        catch (AlreadyExistsException)
        {
            scenarioLogger.LogWarning($"CloudFormation stack '{stackName}' already
exists. Please provide a unique name.");
            var newStackName = PromptUserForStackName();
            return await DeployCloudFormationStack(newStackName,
cloudFormationClient, scenarioLogger);
```

```
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, $"An error occurred while deploying the
CloudFormation stack: {stackName}");
            return false;
        }
    }

    /// <summary>
    /// Waits for the CloudFormation stack to be in the CREATE_COMPLETE state.
    /// </summary>
    /// <param name="stackId">The ID of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>True if the stack was created successfully.</returns>
    private static async Task<bool> WaitForStackCompletion(string stackId,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        int retryCount = 0;
        const int maxRetries = 30;
        const int retryDelay = 10000;

        while (retryCount < maxRetries)
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackId
            };

            var describeStacksResponse = await
cloudFormationClient.DescribeStacksAsync(describeStacksRequest);

            if (describeStacksResponse.Stacks.Count > 0)
            {
                if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_COMPLETE)
                {
                    return true;
                }
                if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_FAILED ||
                    describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.ROLLBACK_COMPLETE)
```

```
                {
                    return false;
                }
            }

            Console.WriteLine("Waiting for CloudFormation stack creation to
complete...");
            await Task.Delay(retryDelay);
            retryCount++;
        }

        scenarioLogger.LogError("Timed out waiting for CloudFormation stack creation
to complete.");
        return false;
    }

    /// <summary>
    /// Gets the outputs from the CloudFormation stack.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>A dictionary of stack outputs.</returns>
    private static async Task<Dictionary<string, string>?> GetStackOutputs(string
stackName, IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics>
scenarioLogger)
    {
        try
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackName
            };

            var response = await
cloudFormationClient.DescribeStacksAsync(describeStacksRequest);

            if (response.Stacks.Count > 0)
            {
                var outputs = new Dictionary<string, string>();
                foreach (var output in response.Stacks[0].Outputs)
                {
                    outputs[output.OutputKey] = output.OutputValue;
                }
```

```
                return outputs;
            }

            return null;
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, $"Failed to get stack outputs for
{stackName}");
            return null;
        }
    }

    /// <summary>
    /// Deletes the CloudFormation stack and waits for confirmation.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>True if the stack was deleted successfully.</returns>
    private static async Task<bool> DeleteCloudFormationStack(string stackName,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        try
        {
            var request = new DeleteStackRequest
            {
                StackName = stackName
            };

            await cloudFormationClient.DeleteStackAsync(request);
            Console.WriteLine($"CloudFormation stack '{stackName}' is being deleted.
This may take a few minutes.");

            bool stackDeleted = await WaitForStackDeletion(stackName,
cloudFormationClient, scenarioLogger);

            if (stackDeleted)
            {
                Console.WriteLine($"CloudFormation stack '{stackName}' has been
deleted.");
                return true;
            }
            else
```

```
            {
                scenarioLogger.LogError($"Failed to delete CloudFormation stack
'{stackName}'.");
                return false;
            }
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, $"An error occurred while deleting the
CloudFormation stack: {stackName}");
            return false;
        }
    }

    /// <summary>
    /// Waits for the stack to be deleted.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>True if the stack was deleted successfully.</returns>
    private static async Task<bool> WaitForStackDeletion(string stackName,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        int retryCount = 0;
        const int maxRetries = 30;
        const int retryDelay = 10000;

        while (retryCount < maxRetries)
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackName
            };

            try
            {
                var describeStacksResponse = await
cloudFormationClient.DescribeStacksAsync(describeStacksRequest);

                if (describeStacksResponse.Stacks.Count == 0 ||
                    describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.DELETE_COMPLETE)
                {
```

```
                    return true;
                }
            }
            catch (AmazonCloudFormationException ex) when (ex.ErrorCode ==
"ValidationError")
            {
                return true;
            }

            Console.WriteLine($"Waiting for CloudFormation stack '{stackName}' to be
deleted...");
            await Task.Delay(retryDelay);
            retryCount++;
        }

        scenarioLogger.LogError($"Timed out waiting for CloudFormation stack
'{stackName}' to be deleted.");
        return false;
    }

    /// <summary>
    /// Helper method to get a yes or no response from the user.
    /// </summary>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null && ynResponse.Equals("y",
StringComparison.InvariantCultureIgnoreCase);
        return response;
    }

    /// <summary>
    /// Prompts the user for a stack name.
    /// </summary>
    private static string PromptUserForStackName()
    {
        if (IsInteractive)
        {
            Console.Write($"Enter a name for the CloudFormation stack (press Enter
for default '{_stackName}'): ");
            string? input = Console.ReadLine();
            if (!string.IsNullOrWhiteSpace(input))
            {
```

```
                var regex = new System.Text.RegularExpressions.Regex("[a-zA-Z][-a-
zA-Z0-9]*");
                if (!regex.IsMatch(input))
                {
                    Console.WriteLine($"Invalid stack name. Using default:
 {_stackName}");
                    return _stackName;
                }
                return input;
            }
        }
        return _stackName;
    }
}
```

Kelas pembungkus untuk metode AWS IoT SDK.

```
/// <summary>
/// Wrapper methods to use Amazon IoT Core with .NET.
/// </summary>
public class IoTWrapper
{
    private readonly IAmazonIoT _amazonIoT;
    private readonly IAmazonIotData _amazonIotData;
    private readonly ILogger<IoTWrapper> _logger;

    /// <summary>
    /// Constructor for the IoT wrapper.
    /// </summary>
    /// <param name="amazonIoT">The injected IoT client.</param>
    /// <param name="amazonIotData">The injected IoT Data client.</param>
    /// <param name="logger">The injected logger.</param>
    public IoTWrapper(IAmazonIoT amazonIoT, IAmazonIotData amazonIotData,
 ILogger<IoTWrapper> logger)
    {
        _amazonIoT = amazonIoT;
        _amazonIotData = amazonIotData;
        _logger = logger;
    }

    /// <summary>
    /// Creates an AWS IoT Thing.
```

```
    /// </summary>
    /// <param name="thingName">The name of the Thing to create.</param>
    /// <returns>The ARN of the Thing created, or null if creation failed.</returns>
    public async Task<string?> CreateThingAsync(string thingName)
    {
        try
        {
            var request = new CreateThingRequest
            {
                ThingName = thingName
            };

            var response = await _amazonIoT.CreateThingAsync(request);
            _logger.LogInformation($"Created Thing {thingName} with ARN
{response.ThingArn}");
            return response.ThingArn;
        }
        catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
        {
            _logger.LogWarning($"Thing {thingName} already exists: {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create Thing {thingName}. Here's why:
{ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Creates a device certificate for AWS IoT.
    /// </summary>
    /// <returns>The certificate details including ARN and certificate PEM, or null
if creation failed.</returns>
    public async Task<(string CertificateArn, string CertificatePem, string
CertificateId)?> CreateKeysAndCertificateAsync()
    {
        try
        {
            var request = new CreateKeysAndCertificateRequest
            {
                SetAsActive = true
            };
```

```
            var response = await _amazonIoT.CreateKeysAndCertificateAsync(request);
            _logger.LogInformation($"Created certificate with ARN
{response.CertificateArn}");
            return (response.CertificateArn, response.CertificatePem,
response.CertificateId);
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create certificate. Here's why:
{ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Attaches a certificate to an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="certificateArn">The ARN of the certificate to attach.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> AttachThingPrincipalAsync(string thingName, string
certificateArn)
    {
        try
        {
            var request = new AttachThingPrincipalRequest
            {
                ThingName = thingName,
                Principal = certificateArn
            };

            await _amazonIoT.AttachThingPrincipalAsync(request);
            _logger.LogInformation($"Attached certificate {certificateArn} to Thing
{thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
```

```
        {
            _logger.LogError($"Cannot attach certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't attach certificate to Thing. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Updates an IoT Thing with attributes.
    /// </summary>
    /// <param name="thingName">The name of the Thing to update.</param>
    /// <param name="attributes">Dictionary of attributes to add.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> UpdateThingAsync(string thingName, Dictionary<string,
string> attributes)
    {
        try
        {
            var request = new UpdateThingRequest
            {
                ThingName = thingName,
                AttributePayload = new AttributePayload
                {
                    Attributes = attributes,
                    Merge = true
                }
            };

            await _amazonIoT.UpdateThingAsync(request);
            _logger.LogInformation($"Updated Thing {thingName} with attributes");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot update Thing - resource not found:
{ex.Message}");
            return false;
        }
```

```
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't update Thing attributes. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Gets the AWS IoT endpoint URL.
    /// </summary>
    /// <returns>The endpoint URL, or null if retrieval failed.</returns>
    public async Task<string?> DescribeEndpointAsync()
    {
        try
        {
            var request = new DescribeEndpointRequest
            {
                EndpointType = "iot:Data-ATS"
            };

            var response = await _amazonIoT.DescribeEndpointAsync(request);
            _logger.LogInformation($"Retrieved endpoint:
{response.EndpointAddress}");
            return response.EndpointAddress;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't describe endpoint. Here's why:
{ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Lists all certificates associated with the account.
    /// </summary>
```

```
    /// <returns>List of certificate information, or empty list if listing failed.</
returns>
    public async Task<List<Certificate>> ListCertificatesAsync()
    {
        try
        {
            var request = new ListCertificatesRequest();
            var response = await _amazonIoT.ListCertificatesAsync(request);

            _logger.LogInformation($"Retrieved {response.Certificates.Count}
 certificates");
            return response.Certificates;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
 {ex.Message}");
            return new List<Certificate>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list certificates. Here's why:
 {ex.Message}");
            return new List<Certificate>();
        }
    }

    /// <summary>
    /// Updates the Thing's shadow with new state information.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="shadowPayload">The shadow payload in JSON format.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> UpdateThingShadowAsync(string thingName, string
 shadowPayload)
    {
        try
        {
            var request = new UpdateThingShadowRequest
            {
                ThingName = thingName,
                Payload = new
 MemoryStream(System.Text.Encoding.UTF8.GetBytes(shadowPayload))
            };
```

```
            await _amazonIotData.UpdateThingShadowAsync(request);
            _logger.LogInformation($"Updated shadow for Thing {thingName}");
            return true;
        }
        catch (Amazon.IotData.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot update Thing shadow - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't update Thing shadow. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Gets the Thing's shadow information.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <returns>The shadow data as a string, or null if retrieval failed.</returns>
    public async Task<string?> GetThingShadowAsync(string thingName)
    {
        try
        {
            var request = new GetThingShadowRequest
            {
                ThingName = thingName
            };

            var response = await _amazonIotData.GetThingShadowAsync(request);
            using var reader = new StreamReader(response.Payload);
            var shadowData = await reader.ReadToEndAsync();

            _logger.LogInformation($"Retrieved shadow for Thing {thingName}");
            return shadowData;
        }
        catch (Amazon.IotData.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot get Thing shadow - resource not found:
{ex.Message}");
```

```
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't get Thing shadow. Here's why:
{ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Creates an IoT topic rule.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <param name="snsTopicArn">The ARN of the SNS topic for the action.</param>
    /// <param name="roleArn">The ARN of the IAM role.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> CreateTopicRuleAsync(string ruleName, string
snsTopicArn, string roleArn)
    {
        try
        {
            var request = new CreateTopicRuleRequest
            {
                RuleName = ruleName,
                TopicRulePayload = new TopicRulePayload
                {
                    Sql = "SELECT * FROM 'topic/subtopic'",
                    Description = $"Rule created by .NET example: {ruleName}",
                    Actions = new List<Amazon.IoT.Model.Action>
                    {
                        new Amazon.IoT.Model.Action
                        {
                            Sns = new SnsAction
                            {
                                TargetArn = snsTopicArn,
                                RoleArn = roleArn
                            }
                        },
                    },
                    RuleDisabled = false
                }
            };
```

```
            await _amazonIoT.CreateTopicRuleAsync(request);
            _logger.LogInformation($"Created IoT rule {ruleName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
        {
            _logger.LogWarning($"Rule {ruleName} already exists: {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create topic rule. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Deletes an IoT topic rule.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteTopicRuleAsync(string ruleName)
    {
        try
        {
            var request = new DeleteTopicRuleRequest
            {
                RuleName = ruleName,
            };

            await _amazonIoT.DeleteTopicRuleAsync(request);
            _logger.LogInformation($"Deleted IoT rule {ruleName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogWarning($"Rule {ruleName} not found: {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete topic rule. Here's why:
{ex.Message}");
```

```
            return false;
        }
    }

    /// <summary>
    /// Lists all IoT topic rules.
    /// </summary>
    /// <returns>List of topic rules, or empty list if listing failed.</returns>
    public async Task<List<TopicRuleListItem>> ListTopicRulesAsync()
    {
        try
        {
            var request = new ListTopicRulesRequest();
            var response = await _amazonIoT.ListTopicRulesAsync(request);

            _logger.LogInformation($"Retrieved {response.Rules.Count} IoT rules");
            return response.Rules;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<TopicRuleListItem>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list topic rules. Here's why:
{ex.Message}");
            return new List<TopicRuleListItem>();
        }
    }

    /// <summary>
    /// Searches for IoT Things using the search index.
    /// </summary>
    /// <param name="queryString">The search query string.</param>
    /// <returns>List of Things that match the search criteria, or empty list if
search failed.</returns>
    public async Task<List<ThingDocument>> SearchIndexAsync(string queryString)
    {
        try
        {
            // First, try to perform the search
            var request = new SearchIndexRequest
```

```
            {
                QueryString = queryString
            };

            var response = await _amazonIoT.SearchIndexAsync(request);
            _logger.LogInformation($"Search found {response.Things.Count} Things");
            return response.Things;
        }
        catch (Amazon.IoT.Model.IndexNotReadyException ex)
        {
            _logger.LogWarning($"Search index not ready, setting up indexing
configuration: {ex.Message}");
            return await SetupIndexAndRetrySearchAsync(queryString);
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
        {
            _logger.LogWarning($"Search index not configured, setting up indexing
configuration: {ex.Message}");
            return await SetupIndexAndRetrySearchAsync(queryString);
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<ThingDocument>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't search index. Here's why: {ex.Message}");
            return new List<ThingDocument>();
        }
    }

    /// <summary>
    /// Sets up the indexing configuration and retries the search after waiting for
the index to be ready.
    /// </summary>
    /// <param name="queryString">The search query string.</param>
    /// <returns>List of Things that match the search criteria, or empty list if
setup/search failed.</returns>
    private async Task<List<ThingDocument>> SetupIndexAndRetrySearchAsync(string
queryString)
    {
```

```
        try
        {
            // Update indexing configuration to REGISTRY mode
            _logger.LogInformation("Setting up IoT search indexing
configuration...");
            await _amazonIoT.UpdateIndexingConfigurationAsync(
                new UpdateIndexingConfigurationRequest()
                {
                    ThingIndexingConfiguration = new ThingIndexingConfiguration()
                    {
                        ThingIndexingMode = ThingIndexingMode.REGISTRY
                    }
                });

            _logger.LogInformation("Indexing configuration updated. Waiting for
index to be ready...");

            // Wait for the index to be set up - this can take some time
            const int maxRetries = 10;
            const int retryDelaySeconds = 10;

            for (int attempt = 1; attempt <= maxRetries; attempt++)
            {
                try
                {
                    _logger.LogInformation($"Waiting for index to be ready (attempt
{attempt}/{maxRetries})...");
                    await Task.Delay(TimeSpan.FromSeconds(retryDelaySeconds));

                    // Try to get the current indexing configuration to see if it's
ready
                    var configResponse = await
_amazonIoT.GetIndexingConfigurationAsync(new GetIndexingConfigurationRequest());
                    if (configResponse.ThingIndexingConfiguration?.ThingIndexingMode
== ThingIndexingMode.REGISTRY)
                    {
                        // Try the search again
                        var request = new SearchIndexRequest
                        {
                            QueryString = queryString
                        };

                        var response = await _amazonIoT.SearchIndexAsync(request);
```

```
                        _logger.LogInformation($"Search found
{response.Things.Count} Things after index setup");
                        return response.Things;
                    }
                }
                catch (Amazon.IoT.Model.IndexNotReadyException)
                {
                    // Index still not ready, continue waiting
                    _logger.LogInformation("Index still not ready, continuing to
wait...");
                    continue;
                }
                catch (Amazon.IoT.Model.InvalidRequestException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
                {
                    // Index still not ready, continue waiting
                    _logger.LogInformation("Index still not ready, continuing to
wait...");
                    continue;
                }
            }

            _logger.LogWarning("Timeout waiting for search index to be ready after
configuration update");
            return new List<ThingDocument>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't set up search index configuration. Here's
why: {ex.Message}");
            return new List<ThingDocument>();
        }
    }

    /// <summary>
    /// Detaches a certificate from an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="certificateArn">The ARN of the certificate to detach.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DetachThingPrincipalAsync(string thingName, string
certificateArn)
    {
        try
```

```
        {
            var request = new DetachThingPrincipalRequest
            {
                ThingName = thingName,
                Principal = certificateArn
            };

            await _amazonIoT.DetachThingPrincipalAsync(request);
            _logger.LogInformation($"Detached certificate {certificateArn} from
Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot detach certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't detach certificate from Thing. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Deletes an IoT certificate.
    /// </summary>
    /// <param name="certificateId">The ID of the certificate to delete.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteCertificateAsync(string certificateId)
    {
        try
        {
            // First, update the certificate to inactive state
            var updateRequest = new UpdateCertificateRequest
            {
                CertificateId = certificateId,
                NewStatus = CertificateStatus.INACTIVE
            };
            await _amazonIoT.UpdateCertificateAsync(updateRequest);

            // Then delete the certificate
```

```
            var deleteRequest = new DeleteCertificateRequest
            {
                CertificateId = certificateId
            };

            await _amazonIoT.DeleteCertificateAsync(deleteRequest);
            _logger.LogInformation($"Deleted certificate {certificateId}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot delete certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete certificate. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Deletes an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing to delete.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteThingAsync(string thingName)
    {
        try
        {
            var request = new DeleteThingRequest
            {
                ThingName = thingName
            };

            await _amazonIoT.DeleteThingAsync(request);
            _logger.LogInformation($"Deleted Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
```

```
            _logger.LogError($"Cannot delete Thing - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete Thing. Here's why: {ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Lists IoT Things with pagination support.
    /// </summary>
    /// <returns>List of Things, or empty list if listing failed.</returns>
    public async Task<List<ThingAttribute>> ListThingsAsync()
    {
        try
        {
            // Use pages of 10.
            var request = new ListThingsRequest()
            {
                MaxResults = 10
            };
            var response = await _amazonIoT.ListThingsAsync(request);

            // Since there is not a built-in paginator, use the NextMarker to
paginate.
            bool hasMoreResults = true;

            var things = new List<ThingAttribute>();
            while (hasMoreResults)
            {
                things.AddRange(response.Things);

                // If NextMarker is not null, there are more results. Get the next
page of results.
                if (!String.IsNullOrEmpty(response.NextMarker))
                {
                    request.Marker = response.NextMarker;
                    response = await _amazonIoT.ListThingsAsync(request);
                }
                else
                    hasMoreResults = false;
```

```
            }

            _logger.LogInformation($"Retrieved {things.Count} Things");
            return things;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
 {ex.Message}");
            return new List<ThingAttribute>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list Things. Here's why: {ex.Message}");
            return new List<ThingAttribute>();
        }
    }

}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk .NET .
  - AttachThingPrincipal
  - CreateKeysAndCertificate
  - CreateThing
  - CreateTopicRule
  - DeleteCertificate
  - DeleteThing
  - DeleteTopicRule
  - DescribeEndpoint
  - DescribeThing
  - DetachThingPrincipal
  - ListCertificates
  - ListThings
  - SearchIndex
  - UpdateIndexingConfiguration
  - UpdateThing

Tindakan

`AttachThingPrincipal`

Contoh kode berikut menunjukkan cara menggunakan`AttachThingPrincipal`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   /// <summary>
   /// Attaches a certificate to an IoT Thing.
   /// </summary>
   /// <param name="thingName">The name of the Thing.</param>
   /// <param name="certificateArn">The ARN of the certificate to attach.</param>
   /// <returns>True if successful, false otherwise.</returns>
   public async Task<bool> AttachThingPrincipalAsync(string thingName, string
certificateArn)
   {
       try
       {
           var request = new AttachThingPrincipalRequest
           {
               ThingName = thingName,
               Principal = certificateArn
           };

           await _amazonIoT.AttachThingPrincipalAsync(request);
           _logger.LogInformation($"Attached certificate {certificateArn} to Thing
{thingName}");
           return true;
       }
       catch (Amazon.IoT.Model.ResourceNotFoundException ex)
       {
           _logger.LogError($"Cannot attach certificate - resource not found:
{ex.Message}");
           return false;
       }
```

```
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't attach certificate to Thing. Here's why:
{ex.Message}");
            return false;
        }
    }
```

- Untuk detail API, lihat AttachThingPrincipaldi Referensi AWS SDK untuk .NET API.

`CreateKeysAndCertificate`

Contoh kode berikut menunjukkan cara menggunakan`CreateKeysAndCertificate`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Creates a device certificate for AWS IoT.
    /// </summary>
    /// <returns>The certificate details including ARN and certificate PEM, or null
if creation failed.</returns>
    public async Task<(string CertificateArn, string CertificatePem, string
CertificateId)?> CreateKeysAndCertificateAsync()
    {
        try
        {
            var request = new CreateKeysAndCertificateRequest
            {
                SetAsActive = true
            };

            var response = await _amazonIoT.CreateKeysAndCertificateAsync(request);
            _logger.LogInformation($"Created certificate with ARN
{response.CertificateArn}");
```

```
            return (response.CertificateArn, response.CertificatePem,
response.CertificateId);
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create certificate. Here's why:
{ex.Message}");
            return null;
        }
    }
```

- Untuk detail API, lihat CreateKeysAndCertificatedi Referensi AWS SDK untuk .NET API.

`CreateThing`

Contoh kode berikut menunjukkan cara menggunakan`CreateThing`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Creates an AWS IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing to create.</param>
    /// <returns>The ARN of the Thing created, or null if creation failed.</returns>
    public async Task<string?> CreateThingAsync(string thingName)
    {
        try
        {
            var request = new CreateThingRequest
```

```
            {
                ThingName = thingName
            };

            var response = await _amazonIoT.CreateThingAsync(request);
            _logger.LogInformation($"Created Thing {thingName} with ARN
{response.ThingArn}");
            return response.ThingArn;
        }
        catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
        {
            _logger.LogWarning($"Thing {thingName} already exists: {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create Thing {thingName}. Here's why:
{ex.Message}");
            return null;
        }
    }
```

- Untuk detail API, lihat CreateThingdi Referensi AWS SDK untuk .NET API.

`CreateTopicRule`

Contoh kode berikut menunjukkan cara menggunakan`CreateTopicRule`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Creates an IoT topic rule.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <param name="snsTopicArn">The ARN of the SNS topic for the action.</param>
```

```
    /// <param name="roleArn">The ARN of the IAM role.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> CreateTopicRuleAsync(string ruleName, string
snsTopicArn, string roleArn)
    {
        try
        {
            var request = new CreateTopicRuleRequest
            {
                RuleName = ruleName,
                TopicRulePayload = new TopicRulePayload
                {
                    Sql = "SELECT * FROM 'topic/subtopic'",
                    Description = $"Rule created by .NET example: {ruleName}",
                    Actions = new List<Amazon.IoT.Model.Action>
                    {
                        new Amazon.IoT.Model.Action
                        {
                            Sns = new SnsAction
                            {
                                TargetArn = snsTopicArn,
                                RoleArn = roleArn
                            }
                        }
                    },
                    RuleDisabled = false
                }
            };

            await _amazonIoT.CreateTopicRuleAsync(request);
            _logger.LogInformation($"Created IoT rule {ruleName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
        {
            _logger.LogWarning($"Rule {ruleName} already exists: {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create topic rule. Here's why:
{ex.Message}");
            return false;
        }
```

```
    }
```

- Untuk detail API, lihat CreateTopicRuledi Referensi AWS SDK untuk .NET API.

`DeleteCertificate`

Contoh kode berikut menunjukkan cara menggunakan`DeleteCertificate`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
/// <summary>
/// Deletes an IoT certificate.
/// </summary>
/// <param name="certificateId">The ID of the certificate to delete.</param>
/// <returns>True if successful, false otherwise.</returns>
public async Task<bool> DeleteCertificateAsync(string certificateId)
{
    try
    {
        // First, update the certificate to inactive state
        var updateRequest = new UpdateCertificateRequest
        {
            CertificateId = certificateId,
            NewStatus = CertificateStatus.INACTIVE
        };
        await _amazonIoT.UpdateCertificateAsync(updateRequest);

        // Then delete the certificate
        var deleteRequest = new DeleteCertificateRequest
        {
            CertificateId = certificateId
        };

        await _amazonIoT.DeleteCertificateAsync(deleteRequest);
        _logger.LogInformation($"Deleted certificate {certificateId}");
```

```
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot delete certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete certificate. Here's why:
{ex.Message}");
            return false;
        }
    }
```

- Untuk detail API, lihat DeleteCertificatedi Referensi AWS SDK untuk .NET API.

`DeleteThing`

Contoh kode berikut menunjukkan cara menggunakan`DeleteThing`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Deletes an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing to delete.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteThingAsync(string thingName)
    {
        try
        {
            var request = new DeleteThingRequest
            {
```

```
                ThingName = thingName
            };

            await _amazonIoT.DeleteThingAsync(request);
            _logger.LogInformation($"Deleted Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot delete Thing - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete Thing. Here's why: {ex.Message}");
            return false;
        }
    }
```

- Untuk detail API, lihat DeleteThingdi Referensi AWS SDK untuk .NET API.

`DescribeEndpoint`

Contoh kode berikut menunjukkan cara menggunakan`DescribeEndpoint`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Gets the AWS IoT endpoint URL.
    /// </summary>
    /// <returns>The endpoint URL, or null if retrieval failed.</returns>
    public async Task<string?> DescribeEndpointAsync()
    {
        try
```

```
        {
            var request = new DescribeEndpointRequest
            {
                EndpointType = "iot:Data-ATS"
            };

            var response = await _amazonIoT.DescribeEndpointAsync(request);
            _logger.LogInformation($"Retrieved endpoint:
{response.EndpointAddress}");
            return response.EndpointAddress;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't describe endpoint. Here's why:
{ex.Message}");
            return null;
        }
    }
```

- Untuk detail API, lihat DescribeEndpointdi Referensi AWS SDK untuk .NET API.

`DetachThingPrincipal`

Contoh kode berikut menunjukkan cara menggunakan`DetachThingPrincipal`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
```

```
    /// Detaches a certificate from an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="certificateArn">The ARN of the certificate to detach.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DetachThingPrincipalAsync(string thingName, string
certificateArn)
    {
        try
        {
            var request = new DetachThingPrincipalRequest
            {
                ThingName = thingName,
                Principal = certificateArn
            };

            await _amazonIoT.DetachThingPrincipalAsync(request);
            _logger.LogInformation($"Detached certificate {certificateArn} from
Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot detach certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't detach certificate from Thing. Here's why:
{ex.Message}");
            return false;
        }
    }
```

- Untuk detail API, lihat DetachThingPrincipaldi Referensi AWS SDK untuk .NET API.

`ListCertificates`

Contoh kode berikut menunjukkan cara menggunakan`ListCertificates`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Lists all certificates associated with the account.
    /// </summary>
    /// <returns>List of certificate information, or empty list if listing failed.</
returns>
    public async Task<List<Certificate>> ListCertificatesAsync()
    {
        try
        {
            var request = new ListCertificatesRequest();
            var response = await _amazonIoT.ListCertificatesAsync(request);

            _logger.LogInformation($"Retrieved {response.Certificates.Count}
 certificates");
            return response.Certificates;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
 {ex.Message}");
            return new List<Certificate>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list certificates. Here's why:
 {ex.Message}");
            return new List<Certificate>();
        }
    }
```

- Untuk detail API, lihat ListCertificatesdi Referensi AWS SDK untuk .NET API.

`ListThings`

Contoh kode berikut menunjukkan cara menggunakan`ListThings`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Lists IoT Things with pagination support.
    /// </summary>
    /// <returns>List of Things, or empty list if listing failed.</returns>
    public async Task<List<ThingAttribute>> ListThingsAsync()
    {
        try
        {
            // Use pages of 10.
            var request = new ListThingsRequest()
            {
                MaxResults = 10
            };
            var response = await _amazonIoT.ListThingsAsync(request);

            // Since there is not a built-in paginator, use the NextMarker to
paginate.
            bool hasMoreResults = true;

            var things = new List<ThingAttribute>();
            while (hasMoreResults)
            {
                things.AddRange(response.Things);

                // If NextMarker is not null, there are more results. Get the next
page of results.
                if (!String.IsNullOrEmpty(response.NextMarker))
                {
                    request.Marker = response.NextMarker;
                    response = await _amazonIoT.ListThingsAsync(request);
                }
```

```
                else
                    hasMoreResults = false;
            }

            _logger.LogInformation($"Retrieved {things.Count} Things");
            return things;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<ThingAttribute>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list Things. Here's why: {ex.Message}");
            return new List<ThingAttribute>();
        }
    }
```

- Untuk detail API, lihat ListThingsdi Referensi AWS SDK untuk .NET API.

`SearchIndex`

Contoh kode berikut menunjukkan cara menggunakan`SearchIndex`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Searches for IoT Things using the search index.
    /// </summary>
    /// <param name="queryString">The search query string.</param>
    /// <returns>List of Things that match the search criteria, or empty list if
search failed.</returns>
    public async Task<List<ThingDocument>> SearchIndexAsync(string queryString)
```

```
    {
        try
        {
            // First, try to perform the search
            var request = new SearchIndexRequest
            {
                QueryString = queryString
            };

            var response = await _amazonIoT.SearchIndexAsync(request);
            _logger.LogInformation($"Search found {response.Things.Count} Things");
            return response.Things;
        }
        catch (Amazon.IoT.Model.IndexNotReadyException ex)
        {
            _logger.LogWarning($"Search index not ready, setting up indexing
configuration: {ex.Message}");
            return await SetupIndexAndRetrySearchAsync(queryString);
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
        {
            _logger.LogWarning($"Search index not configured, setting up indexing
configuration: {ex.Message}");
            return await SetupIndexAndRetrySearchAsync(queryString);
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<ThingDocument>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't search index. Here's why: {ex.Message}");
            return new List<ThingDocument>();
        }
    }

    /// <summary>
    /// Sets up the indexing configuration and retries the search after waiting for
the index to be ready.
    /// </summary>
    /// <param name="queryString">The search query string.</param>
```

```
   /// <returns>List of Things that match the search criteria, or empty list if
setup/search failed.</returns>
   private async Task<List<ThingDocument>> SetupIndexAndRetrySearchAsync(string
queryString)
   {
       try
       {
           // Update indexing configuration to REGISTRY mode
           _logger.LogInformation("Setting up IoT search indexing
configuration...");
           await _amazonIoT.UpdateIndexingConfigurationAsync(
               new UpdateIndexingConfigurationRequest()
               {
                   ThingIndexingConfiguration = new ThingIndexingConfiguration()
                   {
                       ThingIndexingMode = ThingIndexingMode.REGISTRY
                   }
               });

           _logger.LogInformation("Indexing configuration updated. Waiting for
index to be ready...");

           // Wait for the index to be set up - this can take some time
           const int maxRetries = 10;
           const int retryDelaySeconds = 10;

           for (int attempt = 1; attempt <= maxRetries; attempt++)
           {
               try
               {
                   _logger.LogInformation($"Waiting for index to be ready (attempt
{attempt}/{maxRetries})...");
                   await Task.Delay(TimeSpan.FromSeconds(retryDelaySeconds));

                   // Try to get the current indexing configuration to see if it's
ready
                   var configResponse = await
_amazonIoT.GetIndexingConfigurationAsync(new GetIndexingConfigurationRequest());
                   if (configResponse.ThingIndexingConfiguration?.ThingIndexingMode
== ThingIndexingMode.REGISTRY)
                   {
                       // Try the search again
                       var request = new SearchIndexRequest
                       {
```

```
                        QueryString = queryString
                    };

                    var response = await _amazonIoT.SearchIndexAsync(request);
                    _logger.LogInformation($"Search found
{response.Things.Count} Things after index setup");
                    return response.Things;
                }
            }
            catch (Amazon.IoT.Model.IndexNotReadyException)
            {
                // Index still not ready, continue waiting
                _logger.LogInformation("Index still not ready, continuing to
wait...");
                continue;
            }
            catch (Amazon.IoT.Model.InvalidRequestException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
            {
                // Index still not ready, continue waiting
                _logger.LogInformation("Index still not ready, continuing to
wait...");
                continue;
            }
        }

        _logger.LogWarning("Timeout waiting for search index to be ready after
configuration update");
        return new List<ThingDocument>();
    }
    catch (Exception ex)
    {
        _logger.LogError($"Couldn't set up search index configuration. Here's
why: {ex.Message}");
        return new List<ThingDocument>();
    }
}
```

- Untuk detail API, lihat SearchIndexdi Referensi AWS SDK untuk .NET API.

`UpdateThing`

Contoh kode berikut menunjukkan cara menggunakan`UpdateThing`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   /// <summary>
   /// Updates an IoT Thing with attributes.
   /// </summary>
   /// <param name="thingName">The name of the Thing to update.</param>
   /// <param name="attributes">Dictionary of attributes to add.</param>
   /// <returns>True if successful, false otherwise.</returns>
   public async Task<bool> UpdateThingAsync(string thingName, Dictionary<string,
string> attributes)
   {
       try
       {
           var request = new UpdateThingRequest
           {
               ThingName = thingName,
               AttributePayload = new AttributePayload
               {
                   Attributes = attributes,
                   Merge = true
               }
           };

           await _amazonIoT.UpdateThingAsync(request);
           _logger.LogInformation($"Updated Thing {thingName} with attributes");
           return true;
       }
       catch (Amazon.IoT.Model.ResourceNotFoundException ex)
       {
           _logger.LogError($"Cannot update Thing - resource not found:
{ex.Message}");
           return false;
       }
```

```
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't update Thing attributes. Here's why:
{ex.Message}");
            return false;
        }
    }
```

- Untuk detail API, lihat UpdateThingdi Referensi AWS SDK untuk .NET API.

AWS IoT data contoh menggunakan SDK untuk .NET (v4)

Contoh kode berikut menunjukkan cara melakukan tindakan dan mengimplementasikan skenario umum dengan menggunakan AWS SDK untuk .NET (v4) dengan AWS IoT data.

Tindakan merupakan 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

`GetThingShadow`

Contoh kode berikut menunjukkan cara menggunakan`GetThingShadow`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Gets the Thing's shadow information.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <returns>The shadow data as a string, or null if retrieval failed.</returns>
    public async Task<string?> GetThingShadowAsync(string thingName)
    {
        try
        {
            var request = new GetThingShadowRequest
            {
                ThingName = thingName
            };

            var response = await _amazonIotData.GetThingShadowAsync(request);
            using var reader = new StreamReader(response.Payload);
            var shadowData = await reader.ReadToEndAsync();

            _logger.LogInformation($"Retrieved shadow for Thing {thingName}");
            return shadowData;
        }
        catch (Amazon.IotData.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot get Thing shadow - resource not found:
{ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't get Thing shadow. Here's why:
{ex.Message}");
            return null;
        }
    }
```

- Untuk detail API, lihat GetThingShadowdi Referensi AWS SDK untuk .NET API.

`UpdateThingShadow`

Contoh kode berikut menunjukkan cara menggunakan`UpdateThingShadow`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   /// <summary>
   /// Updates the Thing's shadow with new state information.
   /// </summary>
   /// <param name="thingName">The name of the Thing.</param>
   /// <param name="shadowPayload">The shadow payload in JSON format.</param>
   /// <returns>True if successful, false otherwise.</returns>
   public async Task<bool> UpdateThingShadowAsync(string thingName, string
shadowPayload)
   {
       try
       {
           var request = new UpdateThingShadowRequest
           {
               ThingName = thingName,
               Payload = new
MemoryStream(System.Text.Encoding.UTF8.GetBytes(shadowPayload))
           };

           await _amazonIotData.UpdateThingShadowAsync(request);
           _logger.LogInformation($"Updated shadow for Thing {thingName}");
           return true;
       }
       catch (Amazon.IotData.Model.ResourceNotFoundException ex)
       {
           _logger.LogError($"Cannot update Thing shadow - resource not found:
{ex.Message}");
           return false;
       }
       catch (Exception ex)
       {
           _logger.LogError($"Couldn't update Thing shadow. Here's why:
{ex.Message}");
           return false;
       }
```

```
    }
```

- Untuk detail API, lihat UpdateThingShadowdi Referensi AWS SDK untuk .NET API.

Contoh Amazon Redshift menggunakan SDK untuk .NET (v4)

Contoh kode berikut menunjukkan cara melakukan tindakan dan menerapkan skenario umum dengan menggunakan AWS SDK untuk .NET (v4) dengan Amazon Redshift.

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

Tindakan merupakan 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

- Memulai
- Hal-hal mendasar
- Tindakan

Memulai

Halo Amazon Redshift

Contoh kode berikut menunjukkan cara memulai menggunakan Amazon Redshift.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Main method to run the Hello Amazon Redshift example.
    /// </summary>
    /// <param name="args">Command line arguments (not used).</param>
    public static async Task Main(string[] args)
    {
        var redshiftClient = new AmazonRedshiftClient();

        Console.WriteLine("Hello, Amazon Redshift! Let's list available clusters:");

        var clusters = new List<Cluster>();

        try
        {
            // Use pagination to retrieve all clusters.
            var clustersPaginator = redshiftClient.Paginators.DescribeClusters(new
DescribeClustersRequest());

            await foreach (var response in clustersPaginator.Responses)
            {
                if (response.Clusters != null)
                    clusters.AddRange(response.Clusters);
            }

            Console.WriteLine($"{clusters.Count} cluster(s) retrieved.");

            foreach (var cluster in clusters)
            {
                Console.WriteLine($"\t{cluster.ClusterIdentifier} (Status:
{cluster.ClusterStatus})");
            }
        }
        catch (AmazonRedshiftException ex)
        {
            Console.WriteLine($"Couldn't list clusters. Here's why: {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred: {ex.Message}");
        }
    }
```

- Untuk detail API, lihat DescribeClustersdi Referensi AWS SDK untuk .NET API.

Hal-hal mendasar

Pelajari dasar-dasarnya

Contoh kode berikut ini menunjukkan cara untuk melakukan:

- Buat cluster Redshift.
- Daftar database di cluster.
- Buat tabel bernama Movies.
- Isi tabel Film.
- Kueri tabel Film berdasarkan tahun.
- Ubah cluster Redshift.
- Hapus cluster Amazon Redshift.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Buat kelas pembungkus Redshift untuk mengelola operasi.

```
/// <summary>
/// Wrapper class for Amazon Redshift operations.
/// </summary>
public class RedshiftWrapper
{
    private readonly IAmazonRedshift _redshiftClient;
    private readonly IAmazonRedshiftDataAPIService _redshiftDataClient;

    /// <summary>
    /// Constructor for RedshiftWrapper.
    /// </summary>
    /// <param name="redshiftClient">Amazon Redshift client.</param>
```

```
    /// <param name="redshiftDataClient">Amazon Redshift Data API client.</param>
    public RedshiftWrapper(IAmazonRedshift redshiftClient,
IAmazonRedshiftDataAPIService redshiftDataClient)
    {
        _redshiftClient = redshiftClient;
        _redshiftDataClient = redshiftDataClient;
    }

    /// <summary>
    /// Create a new Amazon Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The identifier for the cluster.</param>
    /// <param name="databaseName">The name of the database.</param>
    /// <param name="masterUsername">The master username.</param>
    /// <param name="masterUserPassword">The master user password.</param>
    /// <param name="nodeType">The node type for the cluster.</param>
    /// <returns>The cluster that was created.</returns>
    public async Task<Cluster> CreateClusterAsync(string clusterIdentifier, string
databaseName,
        string masterUsername, string masterUserPassword, string nodeType =
"ra3.large")
    {
        try
        {
            var request = new CreateClusterRequest
            {
                ClusterIdentifier = clusterIdentifier,
                DBName = databaseName,
                MasterUsername = masterUsername,
                MasterUserPassword = masterUserPassword,
                NodeType = nodeType,
                NumberOfNodes = 1,
                ClusterType = "single-node"
            };

            var response = await _redshiftClient.CreateClusterAsync(request);
            Console.WriteLine($"Created cluster {clusterIdentifier}");
            return response.Cluster;
        }
        catch (ClusterAlreadyExistsException ex)
        {
            Console.WriteLine($"Cluster already exists: {ex.Message}");
            throw;
        }
```

```
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't create cluster. Here's why: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Describe Amazon Redshift clusters.
    /// </summary>
    /// <param name="clusterIdentifier">Optional cluster identifier to describe a
specific cluster.</param>
    /// <returns>A list of clusters.</returns>
    public async Task<List<Cluster>> DescribeClustersAsync(string? clusterIdentifier
= null)
    {
        try
        {
            var clusters = new List<Cluster>();
            var request = new DescribeClustersRequest();
            if (!string.IsNullOrEmpty(clusterIdentifier))
            {
                request.ClusterIdentifier = clusterIdentifier;
            }

            var clustersPaginator =
_redshiftClient.Paginators.DescribeClusters(request);
            await foreach (var response in clustersPaginator.Responses)
            {
                if (response.Clusters != null)
                    clusters.AddRange(response.Clusters);
            }

            Console.WriteLine($"{clusters.Count} cluster(s) retrieved.");
            foreach (var cluster in clusters)
            {
                Console.WriteLine($"\t{cluster.ClusterIdentifier} (Status:
{cluster.ClusterStatus})");
            }

            return clusters;
        }
        catch (ClusterNotFoundException ex)
        {
```

```
            Console.WriteLine($"Cluster {clusterIdentifier} not found:
 {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't describe clusters. Here's why:
 {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Modify an Amazon Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The identifier for the cluster.</param>
    /// <param name="preferredMaintenanceWindow">The preferred maintenance window.</
param>
    /// <returns>True if successful.</returns>
    public async Task<bool> ModifyClusterAsync(string clusterIdentifier, string
 preferredMaintenanceWindow)
    {
        try
        {
            var request = new ModifyClusterRequest
            {
                ClusterIdentifier = clusterIdentifier,
                PreferredMaintenanceWindow = preferredMaintenanceWindow
            };

            var response = await _redshiftClient.ModifyClusterAsync(request);
            Console.WriteLine($"The modified cluster was successfully modified and
 has {response.Cluster.PreferredMaintenanceWindow} as the maintenance window");
            return true;
        }
        catch (ClusterNotFoundException ex)
        {
            Console.WriteLine($"Cluster {clusterIdentifier} not found:
 {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't modify cluster. Here's why: {ex.Message}");
```

```
            return false;
        }
    }

    /// <summary>
    /// Delete an Amazon Redshift cluster without a final snapshot.
    /// </summary>
    /// <param name="clusterIdentifier">The identifier for the cluster.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteClusterWithoutSnapshotAsync(string
clusterIdentifier)
    {
        try
        {
            var request = new DeleteClusterRequest
            {
                ClusterIdentifier = clusterIdentifier,
                SkipFinalClusterSnapshot = true
            };

            var response = await _redshiftClient.DeleteClusterAsync(request);
            Console.WriteLine($"The {clusterIdentifier} was deleted");
            return true;
        }
        catch (ClusterNotFoundException ex)
        {
            Console.WriteLine($"Cluster not found: {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't delete cluster. Here's why: {ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// List databases in a Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="dbUser">The database user.</param>
    /// <param name="dbUser">The database name for authentication.</param>
    /// <returns>A list of database names.</returns>
```

```
    public async Task<List<string>> ListDatabasesAsync(string clusterIdentifier,
string dbUser, string databaseName)
    {
        try
        {
            var request = new ListDatabasesRequest
            {
                ClusterIdentifier = clusterIdentifier,
                DbUser = dbUser,
                Database = databaseName
            };

            var response = await _redshiftDataClient.ListDatabasesAsync(request);
            var databases = new List<string>();

            foreach (var database in response.Databases)
            {
                Console.WriteLine($"The database name is : {database}");
                databases.Add(database);
            }

            return databases;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
        {
            Console.WriteLine($"Validation error: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't list databases. Here's why: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Create a table in the Redshift database.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="database">The database name.</param>
    /// <param name="dbUser">The database user.</param>
    /// <returns>The statement ID.</returns>
    public async Task<string> CreateTableAsync(string clusterIdentifier, string
database, string dbUser)
```

```
    {
        try
        {
            var sqlStatement = @"
                CREATE TABLE Movies (
                    id INTEGER PRIMARY KEY,
                    title VARCHAR(250) NOT NULL,
                    year INTEGER NOT NULL
                )";

            var request = new ExecuteStatementRequest
            {
                ClusterIdentifier = clusterIdentifier,
                Database = database,
                DbUser = dbUser,
                Sql = sqlStatement
            };

            var response = await _redshiftDataClient.ExecuteStatementAsync(request);
            await WaitForStatementToCompleteAsync(response.Id);
            Console.WriteLine("Table created: Movies");
            return response.Id;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
        {
            Console.WriteLine($"Validation error: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't create table. Here's why: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Insert a record into the Movies table using parameterized query.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="database">The database name.</param>
    /// <param name="dbUser">The database user.</param>
    /// <param name="id">The movie ID.</param>
    /// <param name="title">The movie title.</param>
    /// <param name="year">The movie year.</param>
```

```
    /// <returns>The statement ID.</returns>
    public async Task<string> InsertMovieAsync(string clusterIdentifier, string
database, string dbUser,
        int id, string title, int year)
    {
        try
        {
            var sqlStatement = "INSERT INTO Movies (id, title, year) VALUES
(:id, :title, :year)";

            var request = new ExecuteStatementRequest
            {
                ClusterIdentifier = clusterIdentifier,
                Database = database,
                DbUser = dbUser,
                Sql = sqlStatement,
                Parameters = new List<SqlParameter>
                {
                    new SqlParameter { Name = "id", Value = id.ToString() },
                    new SqlParameter { Name = "title", Value = title },
                    new SqlParameter { Name = "year", Value = year.ToString() }
                }
            };

            var response = await _redshiftDataClient.ExecuteStatementAsync(request);
            await WaitForStatementToCompleteAsync(response.Id);
            Console.WriteLine($"Inserted: {title} ({year})");
            return response.Id;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
        {
            Console.WriteLine($"Validation error: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't insert movie. Here's why: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Query movies by year using parameterized query.
    /// </summary>
```

```
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="database">The database name.</param>
    /// <param name="dbUser">The database user.</param>
    /// <param name="year">The year to query.</param>
    /// <returns>A list of movie titles.</returns>
    public async Task<List<string>> QueryMoviesByYearAsync(string clusterIdentifier,
string database,
        string dbUser, int year)
    {
        try
        {
            var sqlStatement = "SELECT title FROM Movies WHERE year = :year";

            var request = new ExecuteStatementRequest
            {
                ClusterIdentifier = clusterIdentifier,
                Database = database,
                DbUser = dbUser,
                Sql = sqlStatement,
                Parameters = new List<SqlParameter>
                {
                    new SqlParameter { Name = "year", Value = year.ToString() }
                }
            };

            var response = await _redshiftDataClient.ExecuteStatementAsync(request);
            Console.WriteLine($"The identifier of the statement is {response.Id}");

            await WaitForStatementToCompleteAsync(response.Id);

            var results = await GetStatementResultAsync(response.Id);
            var movieTitles = new List<string>();

            foreach (var row in results)
            {
                if (row.Count > 0)
                {
                    var title = row[0].StringValue;
                    Console.WriteLine($"The Movie title field is {title}");
                    movieTitles.Add(title);
                }
            }

            return movieTitles;
```

```
        }
        catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
        {
            Console.WriteLine($"Validation error: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't query movies. Here's why: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Describe a statement execution.
    /// </summary>
    /// <param name="statementId">The statement ID.</param>
    /// <returns>The statement description.</returns>
    public async Task<DescribeStatementResponse> DescribeStatementAsync(string
statementId)
    {
        try
        {
            var request = new DescribeStatementRequest
            {
                Id = statementId
            };

            var response = await
_redshiftDataClient.DescribeStatementAsync(request);
            return response;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ResourceNotFoundException ex)
        {
            Console.WriteLine($"Statement not found: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't describe statement. Here's why:
{ex.Message}");
            throw;
        }
    }
```

```
    /// <summary>
    /// Get the results of a statement execution.
    /// </summary>
    /// <param name="statementId">The statement ID.</param>
    /// <returns>A list of result rows.</returns>
    public async Task<List<List<Field>>> GetStatementResultAsync(string statementId)
    {
        try
        {
            var request = new GetStatementResultRequest
            {
                Id = statementId
            };

            var response = await
_redshiftDataClient.GetStatementResultAsync(request);
            return response.Records;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ResourceNotFoundException ex)
        {
            Console.WriteLine($"Statement not found: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't get statement result. Here's why:
{ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Wait for a statement to complete execution.
    /// </summary>
    /// <param name="statementId">The statement ID.</param>
    /// <returns>A task representing the asynchronous operation.</returns>
    private async Task WaitForStatementToCompleteAsync(string statementId)
    {
        var status = StatusString.SUBMITTED;
        DescribeStatementResponse? response = null;

        while (status == StatusString.SUBMITTED || status == StatusString.PICKED ||
status == StatusString.STARTED)
```

```
        {
            await Task.Delay(1000); // Wait 1 second
            response = await DescribeStatementAsync(statementId);
            status = response.Status;
            Console.WriteLine($"...{status}");
        }

        if (status == StatusString.FINISHED)
        {
            Console.WriteLine("The statement is finished!");
        }
        else
        {
            var errorMessage = response?.Error ?? "Unknown error";
            Console.WriteLine($"The statement failed with status: {status}");
            Console.WriteLine($"Error message: {errorMessage}");
        }
    }

    /// <summary>
    /// Wait for a cluster to become available.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <returns>A task representing the asynchronous operation.</returns>
    public async Task WaitForClusterAvailableAsync(string clusterIdentifier)
    {
        Console.WriteLine($"Wait until {clusterIdentifier} is available. This may
take a few minutes.");

        var startTime = DateTime.Now;
        var clusters = await DescribeClustersAsync(clusterIdentifier);

        while (clusters[0].ClusterStatus != "available")
        {
            var elapsed = DateTime.Now - startTime;
            Console.WriteLine($"Elapsed Time: {elapsed:mm\\:ss} - Waiting for
cluster...");

            await Task.Delay(5000); // Wait 5 seconds
            clusters = await DescribeClustersAsync(clusterIdentifier);
        }

        var totalElapsed = DateTime.Now - startTime;
```

```
        Console.WriteLine($"Cluster is available! Total Elapsed Time:
 {totalElapsed:mm\\:ss}");
    }
}
```

Jalankan skenario interaktif yang menunjukkan dasar-dasar Redshift.

```
/// <summary>
/// Amazon Redshift Getting Started Scenario.
/// </summary>
public class RedshiftBasics
{
    public static bool IsInteractive = true;
    public static RedshiftWrapper? Wrapper = null;
    public static ILogger logger = null!;
    private static readonly string _moviesFilePath = "../../../../../../resources/
sample_files/movies.json";

    /// <summary>
    /// Main method for the Amazon Redshift Getting Started scenario.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    public static async Task Main(string[] args)
    {
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonRedshift>()
                    .AddAWSService<IAmazonRedshiftDataAPIService>()
                    .AddTransient<RedshiftWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<RedshiftBasics>();

        Wrapper = host.Services.GetRequiredService<RedshiftWrapper>();

        await RunScenarioAsync();
    }

    /// <summary>
    /// Run the complete Amazon Redshift scenario.
```

```
    /// </summary>
    public static async Task RunScenarioAsync()
    {
        // Set all variables to default values
        string userName = "awsuser";
        string userPassword = "AwsUser1000";
        string clusterIdentifier = "redshift-cluster-movies";
        var databaseName = "dev";
        int recordCount = 50;
        int year = 2013;
        try
        {
            Console.WriteLine(

"================================================================================");
            Console.WriteLine("Welcome to the Amazon Redshift SDK Getting Started
scenario.");
            Console.WriteLine(
                "This .NET program demonstrates how to interact with Amazon Redshift
by using the AWS SDK for .NET.");
            Console.WriteLine("Let's get started...");
            Console.WriteLine(

"================================================================================");

            // Step 1: Get user credentials (if interactive)
            if (IsInteractive)
            {
                Console.WriteLine("Please enter a user name for the cluster (default
is awsuser):");
                var userInput = Console.ReadLine();
                if (!string.IsNullOrEmpty(userInput))
                    userName = userInput;


Console.WriteLine("================================================================================
                Console.WriteLine("Please enter a user password for the cluster
(default is AwsUser1000):");
                var passwordInput = Console.ReadLine();
                if (!string.IsNullOrEmpty(passwordInput))
                    userPassword = passwordInput;


Console.WriteLine("================================================================================
```

```
                // Step 2: Get cluster identifier
                Console.WriteLine("Enter a cluster id value (default is redshift-
cluster-movies):");
                var clusterInput = Console.ReadLine();
                if (!string.IsNullOrEmpty(clusterInput))
                    clusterIdentifier = clusterInput;
            }
            else
            {
                Console.WriteLine($"Using default values: userName={userName},
 clusterIdentifier={clusterIdentifier}");
            }

            // Step 3: Create Redshift cluster
            await Wrapper!.CreateClusterAsync(clusterIdentifier, databaseName,
 userName, userPassword);

 Console.WriteLine("=============================================================================

            // Step 4: Wait for cluster to become available

 Console.WriteLine("=============================================================================
            await Wrapper.WaitForClusterAvailableAsync(clusterIdentifier);

 Console.WriteLine("=============================================================================

            // Step 5: List databases

 Console.WriteLine("=============================================================================
            Console.WriteLine($" When you created {clusterIdentifier}, the dev
 database is created by default and used in this scenario.");
            Console.WriteLine(" To create a custom database, you need to have a
 CREATEDB privilege.");
            Console.WriteLine(" For more information, see the documentation here:
 https://docs.aws.amazon.com/redshift/latest/dg/r_CREATE_DATABASE.html.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

 Console.WriteLine("=============================================================================
```

```
Console.WriteLine("=============================================================================
            Console.WriteLine($"List databases in {clusterIdentifier}");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }
            await Wrapper.ListDatabasesAsync(clusterIdentifier, userName,
databaseName);

Console.WriteLine("=============================================================================

            // Step 6: Create Movies table

Console.WriteLine("=============================================================================
            Console.WriteLine("Now you will create a table named Movies.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }
            await Wrapper.CreateTableAsync(clusterIdentifier, databaseName,
userName);

Console.WriteLine("=============================================================================

            // Step 7: Populate the Movies table

Console.WriteLine("=============================================================================
            Console.WriteLine("Populate the Movies table using the Movies.json
file.");

            if (IsInteractive)
            {
                Console.WriteLine("Specify the number of records you would like to
add to the Movies Table.");
                Console.WriteLine("Please enter a value between 50 and 200.");
                Console.Write("Enter a value: ");

                var recordCountInput = Console.ReadLine();
                if (int.TryParse(recordCountInput, out var inputCount) && inputCount
is >= 50 and <= 200)
                {
```

```
                    recordCount = inputCount;
                }
                else
                {
                    Console.WriteLine($"Invalid input. Using default value of
{recordCount}.");
                }
            }
            else
            {
                Console.WriteLine($"Using default record count: {recordCount}");
            }

            await PopulateMoviesTableAsync(clusterIdentifier, databaseName,
userName, recordCount);
            Console.WriteLine($"{recordCount} records were added to the Movies
table.");

Console.WriteLine("================================================================================

            // Step 8 & 9: Query movies by year

Console.WriteLine("================================================================================
            Console.WriteLine("Query the Movies table by year. Enter a value between
2012-2014.");

            if (IsInteractive)
            {
                Console.Write("Enter a year: ");
                var yearInput = Console.ReadLine();
                if (int.TryParse(yearInput, out var inputYear) && inputYear is >=
2012 and <= 2014)
                {
                    year = inputYear;
                }
                else
                {
                    Console.WriteLine($"Invalid input. Using default value of
{year}.");
                }
            }
            else
            {
                Console.WriteLine($"Using default year: {year}");
```

```
            }

            await Wrapper.QueryMoviesByYearAsync(clusterIdentifier, databaseName,
 userName, year);

 Console.WriteLine("=======================================================================

            // Step 10: Modify the cluster

 Console.WriteLine("=======================================================================
            Console.WriteLine("Now you will modify the Redshift cluster.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }
            await Wrapper.ModifyClusterAsync(clusterIdentifier, "wed:07:30-
wed:08:00");

 Console.WriteLine("=======================================================================

            // Step 11 & 12: Delete cluster confirmation

 Console.WriteLine("=======================================================================
            if (IsInteractive)
            {
                Console.WriteLine("Would you like to delete the Amazon Redshift
 cluster? (y/n)");
                var deleteResponse = Console.ReadLine();
                if (deleteResponse?.ToLower() == "y")
                {
                    await
 Wrapper.DeleteClusterWithoutSnapshotAsync(clusterIdentifier);
                }
            }
            else
            {
                Console.WriteLine("Deleting the Amazon Redshift cluster...");
                await Wrapper.DeleteClusterWithoutSnapshotAsync(clusterIdentifier);
            }

 Console.WriteLine("=======================================================================
```

```
Console.WriteLine("=============================================================================
            Console.WriteLine("This concludes the Amazon Redshift SDK Getting
Started scenario.");

Console.WriteLine("=============================================================================
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred during the scenario:
{ex.Message}");
            Console.WriteLine("Deleting the Amazon Redshift cluster...");
            await Wrapper!.DeleteClusterWithoutSnapshotAsync(clusterIdentifier);
            throw;
        }
    }

    /// <summary>
    /// Populate the Movies table with data from the JSON file.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="database">The database name.</param>
    /// <param name="dbUser">The database user.</param>
    /// <param name="recordCount">Number of records to insert.</param>
    private static async Task PopulateMoviesTableAsync(string clusterIdentifier,
string database, string dbUser, int recordCount)
    {
        if (!File.Exists(_moviesFilePath))
        {
            throw new FileNotFoundException($"Required movies data file not found
at: {_moviesFilePath}");
        }

        var jsonContent = await File.ReadAllTextAsync(_moviesFilePath);
        var options = new JsonSerializerOptions
        {
            PropertyNameCaseInsensitive = true
        };
        var movies = JsonSerializer.Deserialize<List<Movie>>(jsonContent, options);

        if (movies == null || movies.Count == 0)
        {
            throw new InvalidOperationException("Failed to parse movies JSON file or
file is empty.");
```

```
        }

        var insertCount = Math.Min(recordCount, movies.Count);

        for (int i = 0; i < insertCount; i++)
        {
            var movie = movies[i];
            await Wrapper!.InsertMovieAsync(clusterIdentifier, database, dbUser, i,
 movie.Title, movie.Year);
        }
    }

    /// <summary>
    /// Movie data model.
    /// </summary>
    private class Movie
    {
        public string Title { get; set; } = string.Empty;
        public int Year { get; set; }
    }
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk .NET .
  - CreateCluster
  - DescribeClusters
  - DescribeStatement
  - ExecuteStatement
  - GetStatementResult
  - ListDatabasesPaginator
  - ModifyCluster

Tindakan

`CreateCluster`

Contoh kode berikut menunjukkan cara menggunakan`CreateCluster`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Create a new Amazon Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The identifier for the cluster.</param>
    /// <param name="databaseName">The name of the database.</param>
    /// <param name="masterUsername">The master username.</param>
    /// <param name="masterUserPassword">The master user password.</param>
    /// <param name="nodeType">The node type for the cluster.</param>
    /// <returns>The cluster that was created.</returns>
    public async Task<Cluster> CreateClusterAsync(string clusterIdentifier, string
databaseName,
        string masterUsername, string masterUserPassword, string nodeType =
"ra3.large")
    {
        try
        {
            var request = new CreateClusterRequest
            {
                ClusterIdentifier = clusterIdentifier,
                DBName = databaseName,
                MasterUsername = masterUsername,
                MasterUserPassword = masterUserPassword,
                NodeType = nodeType,
                NumberOfNodes = 1,
                ClusterType = "single-node"
            };

            var response = await _redshiftClient.CreateClusterAsync(request);
            Console.WriteLine($"Created cluster {clusterIdentifier}");
            return response.Cluster;
        }
        catch (ClusterAlreadyExistsException ex)
        {
            Console.WriteLine($"Cluster already exists: {ex.Message}");
```

```
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't create cluster. Here's why: {ex.Message}");
            throw;
        }
    }
```

- Untuk detail API, lihat CreateClusterdi Referensi AWS SDK untuk .NET API.

`DeleteCluster`

Contoh kode berikut menunjukkan cara menggunakan`DeleteCluster`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Delete an Amazon Redshift cluster without a final snapshot.
    /// </summary>
    /// <param name="clusterIdentifier">The identifier for the cluster.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteClusterWithoutSnapshotAsync(string
clusterIdentifier)
    {
        try
        {
            var request = new DeleteClusterRequest
            {
                ClusterIdentifier = clusterIdentifier,
                SkipFinalClusterSnapshot = true
            };

            var response = await _redshiftClient.DeleteClusterAsync(request);
```

```
            Console.WriteLine($"The {clusterIdentifier} was deleted");
            return true;
        }
        catch (ClusterNotFoundException ex)
        {
            Console.WriteLine($"Cluster not found: {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't delete cluster. Here's why: {ex.Message}");
            return false;
        }
    }
```

- Untuk detail API, lihat DeleteClusterdi Referensi AWS SDK untuk .NET API.

`DescribeClusters`

Contoh kode berikut menunjukkan cara menggunakan`DescribeClusters`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Describe Amazon Redshift clusters.
    /// </summary>
    /// <param name="clusterIdentifier">Optional cluster identifier to describe a
specific cluster.</param>
    /// <returns>A list of clusters.</returns>
    public async Task<List<Cluster>> DescribeClustersAsync(string? clusterIdentifier
= null)
    {
        try
        {
            var clusters = new List<Cluster>();
```

```
            var request = new DescribeClustersRequest();
            if (!string.IsNullOrEmpty(clusterIdentifier))
            {
                request.ClusterIdentifier = clusterIdentifier;
            }

            var clustersPaginator =
_redshiftClient.Paginators.DescribeClusters(request);
            await foreach (var response in clustersPaginator.Responses)
            {
                if (response.Clusters != null)
                    clusters.AddRange(response.Clusters);
            }

            Console.WriteLine($"{clusters.Count} cluster(s) retrieved.");
            foreach (var cluster in clusters)
            {
                Console.WriteLine($"\t{cluster.ClusterIdentifier} (Status:
{cluster.ClusterStatus})");
            }

            return clusters;
        }
        catch (ClusterNotFoundException ex)
        {
            Console.WriteLine($"Cluster {clusterIdentifier} not found:
{ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't describe clusters. Here's why:
{ex.Message}");
            throw;
        }
    }
```

- Untuk detail API, lihat DescribeClustersdi Referensi AWS SDK untuk .NET API.

`DescribeStatement`

Contoh kode berikut menunjukkan cara menggunakan`DescribeStatement`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Describe a statement execution.
    /// </summary>
    /// <param name="statementId">The statement ID.</param>
    /// <returns>The statement description.</returns>
    public async Task<DescribeStatementResponse> DescribeStatementAsync(string
statementId)
    {
        try
        {
            var request = new DescribeStatementRequest
            {
                Id = statementId
            };

            var response = await
_redshiftDataClient.DescribeStatementAsync(request);
            return response;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ResourceNotFoundException ex)
        {
            Console.WriteLine($"Statement not found: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't describe statement. Here's why:
{ex.Message}");
            throw;
        }
    }
```

- Untuk detail API, lihat DescribeStatementdi Referensi AWS SDK untuk .NET API.

`GetStatementResult`

Contoh kode berikut menunjukkan cara menggunakan`GetStatementResult`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Get the results of a statement execution.
    /// </summary>
    /// <param name="statementId">The statement ID.</param>
    /// <returns>A list of result rows.</returns>
    public async Task<List<List<Field>>> GetStatementResultAsync(string statementId)
    {
        try
        {
            var request = new GetStatementResultRequest
            {
                Id = statementId
            };

            var response = await
_redshiftDataClient.GetStatementResultAsync(request);
            return response.Records;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ResourceNotFoundException ex)
        {
            Console.WriteLine($"Statement not found: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't get statement result. Here's why:
{ex.Message}");
            throw;
        }
    }
```

- Untuk detail API, lihat GetStatementResultdi Referensi AWS SDK untuk .NET API.

`ListDatabases`

Contoh kode berikut menunjukkan cara menggunakan`ListDatabases`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// List databases in a Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="dbUser">The database user.</param>
    /// <param name="dbUser">The database name for authentication.</param>
    /// <returns>A list of database names.</returns>
    public async Task<List<string>> ListDatabasesAsync(string clusterIdentifier,
string dbUser, string databaseName)
    {
        try
        {
            var request = new ListDatabasesRequest
            {
                ClusterIdentifier = clusterIdentifier,
                DbUser = dbUser,
                Database = databaseName
            };

            var response = await _redshiftDataClient.ListDatabasesAsync(request);
            var databases = new List<string>();

            foreach (var database in response.Databases)
            {
                Console.WriteLine($"The database name is : {database}");
```

```
                databases.Add(database);
            }

            return databases;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
        {
            Console.WriteLine($"Validation error: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't list databases. Here's why: {ex.Message}");
            throw;
        }
    }
```

- Untuk detail API, lihat ListDatabasesdi Referensi AWS SDK untuk .NET API.

`ModifyCluster`

Contoh kode berikut menunjukkan cara menggunakan`ModifyCluster`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Modify an Amazon Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The identifier for the cluster.</param>
    /// <param name="preferredMaintenanceWindow">The preferred maintenance window.</
param>
    /// <returns>True if successful.</returns>
    public async Task<bool> ModifyClusterAsync(string clusterIdentifier, string
 preferredMaintenanceWindow)
```

```
    {
        try
        {
            var request = new ModifyClusterRequest
            {
                ClusterIdentifier = clusterIdentifier,
                PreferredMaintenanceWindow = preferredMaintenanceWindow
            };

            var response = await _redshiftClient.ModifyClusterAsync(request);
            Console.WriteLine($"The modified cluster was successfully modified and
has {response.Cluster.PreferredMaintenanceWindow} as the maintenance window");
            return true;
        }
        catch (ClusterNotFoundException ex)
        {
            Console.WriteLine($"Cluster {clusterIdentifier} not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't modify cluster. Here's why: {ex.Message}");
            return false;
        }
    }
```

- Untuk detail API, lihat ModifyClusterdi Referensi AWS SDK untuk .NET API.

Contoh Amazon S3 menggunakan SDK untuk .NET (v4)

Contoh kode berikut menunjukkan cara melakukan tindakan dan menerapkan skenario umum dengan menggunakan AWS SDK untuk .NET (v4) dengan Amazon S3.

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

Tindakan merupakan 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

- Memulai
- Hal-hal mendasar
- Tindakan
- Skenario

Memulai

Halo Amazon S3

Contoh kode berikut menunjukkan cara memulai menggunakan Amazon S3.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
/// <summary>
/// Hello Amazon Simple Storage Service
// (Amazon S3) example.
/// </summary>
public class HelloS3
{
    /// <summary>
    /// Main method to run the Hello S3 example.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A Task object.</returns>
    public static async Task Main(string[] args)
```

```
    {
        var s3Client = new AmazonS3Client();

        try
        {
            Console.WriteLine("Hello Amazon S3! Let's list your buckets:");
            Console.WriteLine(new string('-', 80));

            // Use the built-in paginator to list buckets
            var request = new ListBucketsRequest();
            var paginator = s3Client.Paginators.ListBuckets(request);

            var buckets = new List<S3Bucket>();

            await foreach (var response in paginator.Responses)
            {
                buckets.AddRange(response.Buckets);
            }

            if (buckets.Any())
            {
                Console.WriteLine($"Found {buckets.Count} S3 buckets:");
                Console.WriteLine();

                foreach (var bucket in buckets)
                {
                    Console.WriteLine($"- Bucket Name: {bucket.BucketName}");
                    Console.WriteLine($"  Creation Date: {bucket.CreationDate:yyyy-
MM-dd HH:mm:ss UTC}");
                    Console.WriteLine();
                }
            }
            else
            {
                Console.WriteLine("No S3 buckets found in your account.");
            }

            Console.WriteLine("Hello S3 completed successfully.");
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"S3 service error occurred: {ex.Message}");
        }
        catch (Exception ex)
```

```
        {
            Console.WriteLine($"Couldn't list S3 buckets. Here's why:
 {ex.Message}");
        }
    }
}
```

- Untuk detail API, lihat ListBucketsdi Referensi AWS SDK untuk .NET API.

Hal-hal mendasar

Pelajari dasar-dasarnya

Contoh kode berikut ini menunjukkan cara untuk melakukan:

- Membuat bucket dan mengunggah file ke dalamnya.
- Mengunduh objek dari bucket.
- Menyalin objek ke subfolder di bucket.
- Membuat daftar objek dalam bucket.
- Menghapus objek bucket dan bucket tersebut.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Jalankan skenario interaktif yang menunjukkan fitur Amazon S3.

```
public class S3_Basics
{
    public static bool IsInteractive = true;
    public static string BucketName = null!;
    public static string TempFilePath = null!;
    public static S3Wrapper _s3Wrapper = null!;
    public static ILogger<S3_Basics> _logger = null!;
```

```
    public static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonS3>()
                    .AddTransient<S3Wrapper>()
                    .AddLogging(builder => builder.AddConsole()))
            .Build();

        _logger = LoggerFactory.Create(builder => builder.AddConsole())
            .CreateLogger<S3_Basics>();

        _s3Wrapper = host.Services.GetRequiredService<S3Wrapper>();

        var sepBar = new string('-', 45);

        Console.WriteLine(sepBar);
        Console.WriteLine("Amazon Simple Storage Service (Amazon S3) basic");
        Console.WriteLine("procedures. This application will:");
        Console.WriteLine("\n\t1. Create a bucket");
        Console.WriteLine("\n\t2. Upload an object to the new bucket");
        Console.WriteLine("\n\t3. Copy the uploaded object to a folder in the
bucket");
        Console.WriteLine("\n\t4. List the items in the new bucket");
        Console.WriteLine("\n\t5. Delete all the items in the bucket");
        Console.WriteLine("\n\t6. Delete the bucket");
        Console.WriteLine(sepBar);

        await RunScenario(_s3Wrapper, _logger);

        Console.WriteLine(sepBar);
        Console.WriteLine("The Amazon S3 scenario has successfully completed.");
        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Run the S3 Basics scenario with injected dependencies.
    /// </summary>
    /// <param name="s3Wrapper">The S3 wrapper instance.</param>
    /// <param name="scenarioLogger">The logger instance.</param>
    /// <returns>A Task object.</returns>
```

```
    public static async Task RunScenario(S3Wrapper s3Wrapper, ILogger<S3_Basics>
scenarioLogger)
    {
        string bucketName = BucketName;
        string filePath = TempFilePath;
        string keyName = string.Empty;

        var sepBar = new string('-', 45);

        try
        {
            // Create a bucket.
            Console.WriteLine($"\n{sepBar}");
            Console.WriteLine("\nCreate a new Amazon S3 bucket.\n");
            Console.WriteLine(sepBar);

            if (IsInteractive)
            {
                Console.Write("Please enter a name for the new bucket: ");
                bucketName = Console.ReadLine();
            }
            else
            {
                Console.WriteLine($"Using bucket name: {bucketName}");
            }

            var success = await s3Wrapper.CreateBucketAsync(bucketName);
            if (success)
            {
                Console.WriteLine($"Successfully created bucket: {bucketName}.\n");
            }
            else
            {
                Console.WriteLine($"Could not create bucket: {bucketName}.\n");
            }

            Console.WriteLine(sepBar);
            Console.WriteLine("Upload a file to the new bucket.");
            Console.WriteLine(sepBar);

            if (IsInteractive)
            {
                // Get the local path and filename for the file to upload.
                while (string.IsNullOrEmpty(filePath))
```

```
                {
                    Console.Write("Please enter the path and filename of the file to
 upload: ");
                    filePath = Console.ReadLine();

                    // Confirm that the file exists on the local computer.
                    if (!File.Exists(filePath))
                    {
                        Console.WriteLine($"Couldn't find {filePath}. Try again.
\n");
                        filePath = string.Empty;
                    }
                }
            }
            else
            {
                // Use the public variable if set, otherwise create a temp file
                if (!string.IsNullOrEmpty(TempFilePath))
                {
                    filePath = TempFilePath;
                    Console.WriteLine($"Using provided test file: {filePath}");
                }
                else
                {
                    // Create a temporary test file for non-interactive mode
                    filePath = Path.GetTempFileName();
                    var testContent = "This is a test file for S3 basics scenario.
\nGenerated on: " + DateTime.UtcNow.ToString("yyyy-MM-dd HH:mm:ss UTC");
                    await File.WriteAllTextAsync(filePath, testContent);
                    Console.WriteLine($"Created temporary test file: {filePath}");
                }
            }

            // Get the file name from the full path.
            keyName = Path.GetFileName(filePath);

            success = await s3Wrapper.UploadFileAsync(bucketName, keyName,
 filePath);

            if (success)
            {
                Console.WriteLine($"Successfully uploaded {keyName} from {filePath}
 to {bucketName}.\n");
            }
```

```
            else
            {
                Console.WriteLine($"Could not upload {keyName}.\n");
            }

            // Set up download path
            string downloadPath = string.Empty;

            if (IsInteractive)
            {
                // Now get a new location where we can save the file.
                while (string.IsNullOrEmpty(downloadPath))
                {
                    // First get the path to which the file will be downloaded.
                    Console.Write("Please enter the path where the file will be
downloaded: ");
                    downloadPath = Console.ReadLine();

                    // Confirm that the file doesn't already exist on the local
computer.
                    if (File.Exists($"{downloadPath}\\{keyName}"))
                    {
                        Console.WriteLine($"Sorry, the file already exists in that
location.\n");
                        downloadPath = string.Empty;
                    }
                }
            }
            else
            {
                downloadPath = Path.GetTempPath();
                var downloadFile = Path.Combine(downloadPath, keyName);
                if (File.Exists(downloadFile))
                {
                    File.Delete(downloadFile);
                }

                Console.WriteLine($"Using download path: {downloadPath}");
            }

            // Download an object from a bucket.
            success = await s3Wrapper.DownloadObjectFromBucketAsync(bucketName,
keyName, downloadPath);
```

```
            if (success)
            {
                Console.WriteLine($"Successfully downloaded {keyName}.\n");
            }
            else
            {
                Console.WriteLine($"Sorry, could not download {keyName}.\n");
            }

            // Copy the object to a different folder in the bucket.
            string folderName = string.Empty;

            if (IsInteractive)
            {
                while (string.IsNullOrEmpty(folderName))
                {
                    Console.Write("Please enter the name of the folder to copy your
object to: ");
                    folderName = Console.ReadLine();
                }
            }
            else
            {
                folderName = "test-folder";
                Console.WriteLine($"Using folder name: {folderName}");
            }

            await s3Wrapper.CopyObjectInBucketAsync(bucketName, keyName,
folderName);

            // List the objects in the bucket.
            await s3Wrapper.ListBucketContentsAsync(bucketName);

            // Delete the contents of the bucket.
            if (IsInteractive)
            {
                Console.WriteLine("Press <Enter> when you are ready to delete the
bucket contents.");
                _ = Console.ReadLine();
            }

            var deleteContentsSuccess = await
s3Wrapper.DeleteBucketContentsAsync(bucketName);
            if (deleteContentsSuccess)
```

```
            {
                Console.WriteLine($"Successfully deleted contents of {bucketName}.
\n");
            }
            else
            {
                Console.WriteLine($"Sorry, could not delete contents of
 {bucketName}.\n");
            }

            if (IsInteractive)
            {
                // Deleting the bucket too quickly after separately deleting its
 contents can
                // cause an error that the bucket isn't empty. To delete contents
 and bucket in one
                // operation, use AmazonS3Util.DeleteS3BucketWithObjectsAsync
                Console.WriteLine("Press <Enter> when you are ready to delete the
 bucket.");
                _ = Console.ReadLine();
            }
            else
            {
                // Add a small delay for non-interactive mode to ensure objects are
 fully deleted.
                Console.WriteLine("Waiting a moment for objects to be fully
 deleted...");
                await Task.Delay(2000);
            }

            // Delete the bucket.
            var deleteSuccess = await s3Wrapper.DeleteBucketAsync(bucketName);
            if (deleteSuccess)
            {
                Console.WriteLine($"Successfully deleted {bucketName}.\n");
            }
            else
            {
                Console.WriteLine($"Sorry, could not delete {bucketName}.\n");
            }

            // Clean up temporary files in non-interactive mode
            if (!IsInteractive)
            {
```

```
                try
                {
                    if (File.Exists(filePath))
                    {
                        File.Delete(filePath);
                        Console.WriteLine("Cleaned up temporary test file.");
                    }

                    var downloadFile = Path.Combine(downloadPath, keyName);
                    if (File.Exists(downloadFile))
                    {
                        File.Delete(downloadFile);
                        Console.WriteLine("Cleaned up downloaded test file.");
                    }
                }
                catch (Exception ex)
                {
                    scenarioLogger.LogWarning(ex, "Failed to clean up temporary
files.");
                }
            }
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, "An error occurred during the S3 scenario
execution.");

            // Clean up on error - delete bucket if it exists
            try
            {
                if (!string.IsNullOrEmpty(bucketName))
                {
                    await s3Wrapper.DeleteBucketContentsAsync(bucketName);
                    await s3Wrapper.DeleteBucketAsync(bucketName);
                }
            }
            catch (Exception cleanupEx)
            {
                scenarioLogger.LogError(cleanupEx, "Error during cleanup.");
            }

            // Clean up temporary files in non-interactive mode
            if (!IsInteractive)
            {
```

```
                try
                {
                    if (!string.IsNullOrEmpty(filePath) && File.Exists(filePath))
                    {
                        File.Delete(filePath);
                    }
                }
                catch (Exception fileCleanupEx)
                {
                    scenarioLogger.LogWarning(fileCleanupEx, "Failed to clean up
 temporary files during error handling.");
                }
            }

            throw;
        }
    }
}
```

Kelas pembungkus untuk metode Amazon S3 SDK.

```
using Amazon.S3;
using Amazon.S3.Model;

namespace S3_Actions;

/// <summary>
/// This class contains all of the methods for working with Amazon Simple
/// Storage Service (Amazon S3) buckets.
/// </summary>
public class S3Wrapper
{
    private readonly IAmazonS3 _amazonS3;

    /// <summary>
    /// Initializes a new instance of the <see cref="S3Wrapper"/> class.
    /// </summary>
    /// <param name="amazonS3">An initialized Amazon S3 client object.</param>
    public S3Wrapper(IAmazonS3 amazonS3)
    {
        _amazonS3 = amazonS3;
```

```
    }


    /// <summary>
    /// Shows how to create a new Amazon S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name of the bucket to create.</param>
    /// <returns>A boolean value representing the success or failure of
    /// the bucket creation process.</returns>
    public async Task<bool> CreateBucketAsync(string bucketName)
    {
        try
        {
            var request = new PutBucketRequest
            {
                BucketName = bucketName,
                UseClientRegion = true,
            };

            var response = await _amazonS3.PutBucketAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error creating bucket: '{ex.Message}'");
            return false;
        }
    }



    /// <summary>
    /// Shows how to upload a file from the local computer to an Amazon S3
    /// bucket.
    /// </summary>
    /// <param name="bucketName">The Amazon S3 bucket to which the object
    /// will be uploaded.</param>
    /// <param name="objectName">The object to upload.</param>
    /// <param name="filePath">The path, including file name, of the object
    /// on the local computer to upload.</param>
    /// <returns>A boolean value indicating the success or failure of the
    /// upload procedure.</returns>
    public async Task<bool> UploadFileAsync(
        string bucketName,
```

```
        string objectName,
        string filePath)
    {
        try
        {
            var request = new PutObjectRequest
            {
                BucketName = bucketName,
                Key = objectName,
                FilePath = filePath,
            };

            var response = await _amazonS3.PutObjectAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error uploading {objectName}: {ex.Message}");
            return false;
        }
    }



    /// <summary>
    /// Shows how to download an object from an Amazon S3 bucket to the
    /// local computer.
    /// </summary>
    /// <param name="bucketName">The name of the bucket where the object is
    /// currently stored.</param>
    /// <param name="objectName">The name of the object to download.</param>
    /// <param name="filePath">The path, including filename, where the
    /// downloaded object will be stored.</param>
    /// <returns>A boolean value indicating the success or failure of the
    /// download process.</returns>
    public async Task<bool> DownloadObjectFromBucketAsync(
        string bucketName,
        string objectName,
        string filePath)
    {
        var request = new GetObjectRequest
        {
            BucketName = bucketName,
            Key = objectName,
```

```
        };

        using GetObjectResponse response = await _amazonS3.GetObjectAsync(request);

        try
        {
            // Save object to local file
            await response.WriteResponseStreamToFileAsync($"{filePath}\
\{objectName}", true, CancellationToken.None);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error saving {objectName}: {ex.Message}");
            return false;
        }
    }



    /// <summary>
    /// Copies an object in an Amazon S3 bucket to a folder within the
    /// same bucket.
    /// </summary>
    /// <param name="bucketName">The name of the Amazon S3 bucket where the
    /// object to copy is located.</param>
    /// <param name="objectName">The object to be copied.</param>
    /// <param name="folderName">The folder to which the object will
    /// be copied.</param>
    /// <returns>A boolean value that indicates the success or failure of
    /// the copy operation.</returns>
    public async Task<bool> CopyObjectInBucketAsync(
        string bucketName,
        string objectName,
        string folderName)
    {
        try
        {
            var request = new CopyObjectRequest
            {
                SourceBucket = bucketName,
                SourceKey = objectName,
                DestinationBucket = bucketName,
                DestinationKey = $"{folderName}\\{objectName}",
```

```
            };
            var response = await _amazonS3.CopyObjectAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error copying object: '{ex.Message}'");
            return false;
        }
    }



    /// <summary>
    /// Shows how to list the objects in an Amazon S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name of the bucket for which to list.
    /// <param name="printList">True to print out the list.
    /// <returns>The collection of objects.</returns>
    public async Task<List<S3Object>?> ListBucketContentsAsync(string bucketName,
bool printList = true)
    {
        try
        {
            var request = new ListObjectsV2Request
            {
                BucketName = bucketName,
                MaxKeys = 5,
            };

            if (printList)
            {
                Console.WriteLine("--------------------------------------");
                Console.WriteLine($"Listing the contents of {bucketName}:");
                Console.WriteLine("--------------------------------------");
            }

            var listObjectsV2Paginator = _amazonS3.Paginators.ListObjectsV2(new
ListObjectsV2Request
            {
                BucketName = bucketName,
            });
            var s3Objects = new List<S3Object>();
            await foreach (var response in listObjectsV2Paginator.Responses)
```

```
            {
                if (response.S3Objects != null)
                {
                    s3Objects.AddRange(response.S3Objects);
                }
            }

            if (printList)
            {
                Console.WriteLine($"Number of Objects: {s3Objects.Count}");
                foreach (var entry in s3Objects)
                {
                    Console.WriteLine($"Key = {entry.Key} Size = {entry.Size}");
                }
            }

            return s3Objects;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error encountered on server. Message:'{ex.Message}'
getting list of objects.");
            return null;
        }
    }



    /// <summary>
    /// Delete all of the objects stored in an existing Amazon S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name of the bucket from which the
    /// contents will be deleted.</param>
    /// <returns>A boolean value that represents the success or failure of
    /// deleting all of the objects in the bucket.</returns>
    public async Task<bool> DeleteBucketContentsAsync(string bucketName)
    {
        // Iterate over the contents of the bucket and delete all objects.
        try
        {
            // Delete all objects in the bucket.
            var deleteList = await ListBucketContentsAsync(bucketName, false);
            if (deleteList != null && deleteList.Any())
            {
```

```
                await _amazonS3.DeleteObjectsAsync(new DeleteObjectsRequest()
                {
                    BucketName = bucketName,
                    Objects = deleteList.Select(o => new KeyVersion { Key =
 o.Key }).ToList(),
                });
            }

            return true;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error deleting objects: {ex.Message}");
            return false;
        }
    }



    /// <summary>
    /// Shows how to delete an Amazon S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name of the Amazon S3 bucket to delete.</param>
    /// <returns>A boolean value that represents the success or failure of
    /// the delete operation.</returns>
    public async Task<bool> DeleteBucketAsync(string bucketName)
    {
        try
        {
            var request = new DeleteBucketRequest { BucketName = bucketName, };

            await _amazonS3.DeleteBucketAsync(request);
            return true;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error deleting bucket: {ex.Message}");
            return false;
        }
    }

}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk .NET .
  - CopyObject
  - CreateBucket
  - DeleteBucket
  - DeleteObjects
  - GetObject
  - ListObjectsV2
  - PutObject

Tindakan

`CopyObject`

Contoh kode berikut menunjukkan cara menggunakan`CopyObject`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
/// <summary>
/// Copies an object in an Amazon S3 bucket to a folder within the
/// same bucket.
/// </summary>
/// <param name="bucketName">The name of the Amazon S3 bucket where the
/// object to copy is located.</param>
/// <param name="objectName">The object to be copied.</param>
/// <param name="folderName">The folder to which the object will
/// be copied.</param>
/// <returns>A boolean value that indicates the success or failure of
/// the copy operation.</returns>
public async Task<bool> CopyObjectInBucketAsync(
```

```
        string bucketName,
        string objectName,
        string folderName)
    {
        try
        {
            var request = new CopyObjectRequest
            {
                SourceBucket = bucketName,
                SourceKey = objectName,
                DestinationBucket = bucketName,
                DestinationKey = $"{folderName}\\{objectName}",
            };
            var response = await _amazonS3.CopyObjectAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error copying object: '{ex.Message}'");
            return false;
        }
    }
```

- Untuk detail API, lihat CopyObjectdi Referensi AWS SDK untuk .NET API.

`CreateBucket`

Contoh kode berikut menunjukkan cara menggunakan`CreateBucket`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Shows how to create a new Amazon S3 bucket.
```

```
    /// </summary>
    /// <param name="bucketName">The name of the bucket to create.</param>
    /// <returns>A boolean value representing the success or failure of
    /// the bucket creation process.</returns>
    public async Task<bool> CreateBucketAsync(string bucketName)
    {
        try
        {
            var request = new PutBucketRequest
            {
                BucketName = bucketName,
                UseClientRegion = true,
            };

            var response = await _amazonS3.PutBucketAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error creating bucket: '{ex.Message}'");
            return false;
        }
    }
```

- Untuk detail API, lihat CreateBucketdi Referensi AWS SDK untuk .NET API.

`CreatePresignedPost`

Contoh kode berikut menunjukkan cara menggunakan`CreatePresignedPost`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Buat URL POST yang telah ditentukan sebelumnya.

```
    /// <summary>
    /// Create a presigned POST URL with conditions.
    /// </summary>
    /// <param name="s3Client">The Amazon S3 client.</param>
    /// <param name="bucketName">The name of the bucket.</param>
    /// <param name="objectKey">The object key (path) where the uploaded file will
 be stored.</param>
    /// <param name="expires">When the presigned URL expires.</param>
    /// <param name="fields">Dictionary of fields to add to the form.</param>
    /// <param name="conditions">List of conditions to apply.</param>
    /// <returns>A CreatePresignedPostResponse object with URL and form fields.</
returns>
    public async Task<CreatePresignedPostResponse> CreatePresignedPostAsync(
        IAmazonS3 s3Client,
        string bucketName,
        string objectKey,
        DateTime expires,
        Dictionary<string, string>? fields = null,
        List<S3PostCondition>? conditions = null)
    {
        var request = new CreatePresignedPostRequest
        {
            BucketName = bucketName,
            Key = objectKey,
            Expires = expires
        };

        // Add custom fields if provided
        if (fields != null)
        {
            foreach (var field in fields)
            {
                request.Fields.Add(field.Key, field.Value);
            }
        }

        // Add conditions if provided
        if (conditions != null)
        {
            foreach (var condition in conditions)
            {
                request.Conditions.Add(condition);
            }
```

```
        }

        return await s3Client.CreatePresignedPostAsync(request);
    }
```

- Untuk detail API, lihat CreatePresignedPostdi Referensi AWS SDK untuk .NET API.

`DeleteBucket`

Contoh kode berikut menunjukkan cara menggunakan`DeleteBucket`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
/// <summary>
/// Shows how to delete an Amazon S3 bucket.
/// </summary>
/// <param name="bucketName">The name of the Amazon S3 bucket to delete.</param>
/// <returns>A boolean value that represents the success or failure of
/// the delete operation.</returns>
public async Task<bool> DeleteBucketAsync(string bucketName)
{
    try
    {
        var request = new DeleteBucketRequest { BucketName = bucketName, };

        await _amazonS3.DeleteBucketAsync(request);
        return true;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Error deleting bucket: {ex.Message}");
        return false;
    }
}
```

- Untuk detail API, lihat DeleteBucketdi Referensi AWS SDK untuk .NET API.

`DeleteObjects`

Contoh kode berikut menunjukkan cara menggunakan`DeleteObjects`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Delete all of the objects stored in an existing Amazon S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name of the bucket from which the
    /// contents will be deleted.</param>
    /// <returns>A boolean value that represents the success or failure of
    /// deleting all of the objects in the bucket.</returns>
    public async Task<bool> DeleteBucketContentsAsync(string bucketName)
    {
        // Iterate over the contents of the bucket and delete all objects.
        try
        {
            // Delete all objects in the bucket.
            var deleteList = await ListBucketContentsAsync(bucketName, false);
            if (deleteList != null && deleteList.Any())
            {
                await _amazonS3.DeleteObjectsAsync(new DeleteObjectsRequest()
                {
                    BucketName = bucketName,
                    Objects = deleteList.Select(o => new KeyVersion { Key =
o.Key }).ToList(),
                });
            }
```

```
            return true;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error deleting objects: {ex.Message}");
            return false;
        }
    }
```

- Untuk detail API, lihat DeleteObjectsdi Referensi AWS SDK untuk .NET API.

`GetObject`

Contoh kode berikut menunjukkan cara menggunakan`GetObject`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
/// <summary>
/// Shows how to download an object from an Amazon S3 bucket to the
/// local computer.
/// </summary>
/// <param name="bucketName">The name of the bucket where the object is
/// currently stored.</param>
/// <param name="objectName">The name of the object to download.</param>
/// <param name="filePath">The path, including filename, where the
/// downloaded object will be stored.</param>
/// <returns>A boolean value indicating the success or failure of the
/// download process.</returns>
public async Task<bool> DownloadObjectFromBucketAsync(
    string bucketName,
    string objectName,
    string filePath)
{
```

```
        var request = new GetObjectRequest
        {
            BucketName = bucketName,
            Key = objectName,
        };

        using GetObjectResponse response = await _amazonS3.GetObjectAsync(request);

        try
        {
            // Save object to local file
            await response.WriteResponseStreamToFileAsync($"{filePath}\
\{objectName}", true, CancellationToken.None);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error saving {objectName}: {ex.Message}");
            return false;
        }
    }
```

- Untuk detail API, lihat GetObjectdi Referensi AWS SDK untuk .NET API.

`ListObjectsV2`

Contoh kode berikut menunjukkan cara menggunakan`ListObjectsV2`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Shows how to list the objects in an Amazon S3 bucket.
    /// </summary>
```

```
    /// <param name="bucketName">The name of the bucket for which to list.
    /// <param name="printList">True to print out the list.
    /// <returns>The collection of objects.</returns>
    public async Task<List<S3Object>?> ListBucketContentsAsync(string bucketName,
bool printList = true)
    {
        try
        {
            var request = new ListObjectsV2Request
            {
                BucketName = bucketName,
                MaxKeys = 5,
            };

            if (printList)
            {
                Console.WriteLine("--------------------------------------");
                Console.WriteLine($"Listing the contents of {bucketName}:");
                Console.WriteLine("--------------------------------------");
            }

            var listObjectsV2Paginator = _amazonS3.Paginators.ListObjectsV2(new
ListObjectsV2Request
            {
                BucketName = bucketName,
            });
            var s3Objects = new List<S3Object>();
            await foreach (var response in listObjectsV2Paginator.Responses)
            {
                if (response.S3Objects != null)
                {
                    s3Objects.AddRange(response.S3Objects);
                }
            }

            if (printList)
            {
                Console.WriteLine($"Number of Objects: {s3Objects.Count}");
                foreach (var entry in s3Objects)
                {
                    Console.WriteLine($"Key = {entry.Key} Size = {entry.Size}");
                }
            }
```

```
            return s3Objects;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error encountered on server. Message:'{ex.Message}'
getting list of objects.");
            return null;
        }
    }
```

- Untuk detail API, lihat ListObjectsV2 di Referensi AWS SDK untuk .NET API.

`PutObject`

Contoh kode berikut menunjukkan cara menggunakan`PutObject`.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Shows how to upload a file from the local computer to an Amazon S3
    /// bucket.
    /// </summary>
    /// <param name="bucketName">The Amazon S3 bucket to which the object
    /// will be uploaded.</param>
    /// <param name="objectName">The object to upload.</param>
    /// <param name="filePath">The path, including file name, of the object
    /// on the local computer to upload.</param>
    /// <returns>A boolean value indicating the success or failure of the
    /// upload procedure.</returns>
    public async Task<bool> UploadFileAsync(
        string bucketName,
        string objectName,
        string filePath)
```

```
    {
        try
        {
            var request = new PutObjectRequest
            {
                BucketName = bucketName,
                Key = objectName,
                FilePath = filePath,
            };

            var response = await _amazonS3.PutObjectAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error uploading {objectName}: {ex.Message}");
            return false;
        }
    }
```

- Untuk detail API, lihat PutObjectdi Referensi AWS SDK untuk .NET API.

Skenario

Membuat URL yang telah ditetapkan sebelumnya

Contoh kode berikut menunjukkan cara membuat URL presigned untuk Amazon S3 dan mengunggah objek.

SDK untuk .NET (v4)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Buat dan gunakan POST presigned URLs untuk upload browser langsung.

```
/// <summary>
```

```
/// Scenario demonstrating the complete workflow for presigned POST URLs:
/// 1. Create an S3 bucket
/// 2. Create a presigned POST URL
/// 3. Upload a file using the presigned POST URL
/// 4. Clean up resources
/// </summary>
public class CreatePresignedPostBasics
{
    public static ILogger<CreatePresignedPostBasics> _logger = null!;
    public static S3Wrapper _s3Wrapper = null!;
    public static UiMethods _uiMethods = null!;
    public static IHttpClientFactory _httpClientFactory = null!;
    public static bool _isInteractive = true;
    public static string? _bucketName;
    public static string? _objectKey;

    /// <summary>
    /// Set up the services and logging.
    /// </summary>
    /// <param name="host">The IHost instance.</param>
    public static void SetUpServices(IHost host)
    {
        var loggerFactory = LoggerFactory.Create(builder =>
        {
            builder.AddConsole();
        });
        _logger = new Logger<CreatePresignedPostBasics>(loggerFactory);

        _s3Wrapper = host.Services.GetRequiredService<S3Wrapper>();
        _httpClientFactory = host.Services.GetRequiredService<IHttpClientFactory>();
        _uiMethods = new UiMethods();
    }

    /// <summary>
    /// Perform the actions defined for the Amazon S3 Presigned POST scenario.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A Task object.</returns>
    public static async Task Main(string[] args)
    {
        _isInteractive = !args.Contains("--non-interactive");

        // Set up dependency injection for Amazon S3
```

```
        using var host =
Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonS3>()
                    .AddTransient<S3Wrapper>()
                    .AddHttpClient()
            )
            .Build();

        SetUpServices(host);

        try
        {
            // Display overview
            _uiMethods.DisplayOverview();
            _uiMethods.PressEnter(_isInteractive);

            // Step 1: Create bucket
            await CreateBucketAsync();
            _uiMethods.PressEnter(_isInteractive);

            // Step 2: Create presigned URL
            _uiMethods.DisplayTitle("Step 2: Create presigned POST URL");
            var response = await CreatePresignedPostAsync();
            _uiMethods.PressEnter(_isInteractive);

            // Step 3: Display URL and fields
            _uiMethods.DisplayTitle("Step 3: Presigned POST URL details");
            DisplayPresignedPostFields(response);
            _uiMethods.PressEnter(_isInteractive);

            // Step 4: Upload file
            _uiMethods.DisplayTitle("Step 4: Upload test file using presigned POST
URL");
            await UploadFileAsync(response);
            _uiMethods.PressEnter(_isInteractive);

            // Step 5: Verify file exists
            await VerifyFileExistsAsync();
            _uiMethods.PressEnter(_isInteractive);

            // Step 6: Cleanup
            _uiMethods.DisplayTitle("Step 6: Clean up resources");
            await CleanupAsync();
```

```
            _uiMethods.DisplayTitle("S3 Presigned POST Scenario completed
 successfully!");
            _uiMethods.PressEnter(_isInteractive);
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "Error in scenario");
            Console.WriteLine($"Error: {ex.Message}");

            // Attempt cleanup if there was an error
            if (!string.IsNullOrEmpty(_bucketName))
            {
                _uiMethods.DisplayTitle("Cleaning up resources after error");
                await _s3Wrapper.DeleteBucketAsync(_bucketName);
                Console.WriteLine($"Cleaned up bucket: {_bucketName}");
            }
        }
    }

    /// <summary>
    /// Create an S3 bucket for the scenario.
    /// </summary>
    private static async Task CreateBucketAsync()
    {
        _uiMethods.DisplayTitle("Step 1: Create an S3 bucket");

        // Generate a default bucket name for the scenario
        var defaultBucketName = $"presigned-post-demo-
{DateTime.Now:yyyyMMddHHmmss}".ToLower();

        // Prompt user for bucket name or use default in non-interactive mode
        _bucketName = _uiMethods.GetUserInput(
            $"Enter S3 bucket name (or press Enter for '{defaultBucketName}'): ",
            defaultBucketName,
            _isInteractive);

        // Basic validation to ensure bucket name is not empty
        if (string.IsNullOrWhiteSpace(_bucketName))
        {
            _bucketName = defaultBucketName;
        }

        Console.WriteLine($"Creating bucket: {_bucketName}");
```

```
        await _s3Wrapper.CreateBucketAsync(_bucketName);

        Console.WriteLine($"Successfully created bucket: {_bucketName}");
    }


    /// <summary>
    /// Create a presigned POST URL.
    /// </summary>
    private static async Task<CreatePresignedPostResponse>
 CreatePresignedPostAsync()
    {
        _objectKey = "example-upload.txt";
        var expiration = DateTime.UtcNow.AddMinutes(10); // Short expiration for the
 demo

        Console.WriteLine($"Creating presigned POST URL for {_bucketName}/
{_objectKey}");
        Console.WriteLine($"Expiration: {expiration} UTC");

        var s3Client = _s3Wrapper.GetS3Client();

        var response = await _s3Wrapper.CreatePresignedPostAsync(
            s3Client, _bucketName!, _objectKey, expiration);

        Console.WriteLine("Successfully created presigned POST URL");
        return response;
    }

    /// <summary>
    /// Upload a file using the presigned POST URL.
    /// </summary>
    private static async Task UploadFileAsync(CreatePresignedPostResponse response)
    {

        // Create a temporary test file to upload
        string testFilePath = Path.GetTempFileName();
        string testContent = "This is a test file for the S3 presigned POST
 scenario.";

        await File.WriteAllTextAsync(testFilePath, testContent);
        Console.WriteLine($"Created test file at: {testFilePath}");
```

```
        // Upload the file using the presigned POST URL
        Console.WriteLine("\nUploading file using the presigned POST URL...");
        var uploadResult = await UploadFileWithPresignedPostAsync(response,
testFilePath);

        // Display the upload result
        if (uploadResult.Success)
        {
            Console.WriteLine($"Upload successful! Status code:
{uploadResult.StatusCode}");
        }
        else
        {
            Console.WriteLine($"Upload failed with status code:
{uploadResult.StatusCode}");
            Console.WriteLine($"Error: {uploadResult.Response}");
            throw new Exception("File upload failed");
        }

        // Clean up the temporary file
        File.Delete(testFilePath);
        Console.WriteLine("Temporary file deleted");
    }

    /// <summary>
    /// Helper method to upload a file using a presigned POST URL.
    /// </summary>
    private static async Task<(bool Success, HttpStatusCode StatusCode, string
Response)> UploadFileWithPresignedPostAsync(
        CreatePresignedPostResponse response,
        string filePath)
    {
        try
        {
            _logger.LogInformation("Uploading file {filePath} using presigned POST
URL", filePath);

            using var httpClient = _httpClientFactory.CreateClient();
            using var formContent = new MultipartFormDataContent();

            // Add all the fields from the presigned POST response
            foreach (var field in response.Fields)
            {
                formContent.Add(new StringContent(field.Value), field.Key);
```

```
            }

            // Add the file content
            var fileStream = File.OpenRead(filePath);
            var fileName = Path.GetFileName(filePath);
            var fileContent = new StreamContent(fileStream);
            fileContent.Headers.ContentType = new MediaTypeHeaderValue("text/
plain");
            formContent.Add(fileContent, "file", fileName);

            // Send the POST request
            var httpResponse = await httpClient.PostAsync(response.Url,
 formContent);
            var responseContent = await httpResponse.Content.ReadAsStringAsync();

            // Log and return the result
            _logger.LogInformation("Upload completed with status code {statusCode}",
 httpResponse.StatusCode);

            return (httpResponse.IsSuccessStatusCode, httpResponse.StatusCode,
 responseContent);
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "Error uploading file");
            return (false, HttpStatusCode.InternalServerError, ex.Message);
        }
    }

    /// <summary>
    /// Verify that the uploaded file exists in the S3 bucket.
    /// </summary>
    private static async Task VerifyFileExistsAsync()
    {
        _uiMethods.DisplayTitle("Step 5: Verify uploaded file exists");

        Console.WriteLine($"Checking if file exists at {_bucketName}/
{_objectKey}...");

        try
        {
            var metadata = await _s3Wrapper.GetObjectMetadataAsync(_bucketName!,
 _objectKey!);
```

```
            Console.WriteLine($"File verification successful! File exists in the
bucket.");
            Console.WriteLine($"File size: {metadata.ContentLength} bytes");
            Console.WriteLine($"File type: {metadata.Headers.ContentType}");
            Console.WriteLine($"Last modified: {metadata.LastModified}");
        }
        catch (AmazonS3Exception ex) when (ex.StatusCode ==
System.Net.HttpStatusCode.NotFound)
        {
            Console.WriteLine($"Error: File was not found in the bucket.");
            throw;
        }
    }

    private static void DisplayPresignedPostFields(CreatePresignedPostResponse
response)
    {
        Console.WriteLine($"Presigned POST URL: {response.Url}");
        Console.WriteLine("Form fields to include:");

        foreach (var field in response.Fields)
        {
            Console.WriteLine($"  {field.Key}: {field.Value}");
        }
    }

    /// <summary>
    /// Clean up resources created by the scenario.
    /// </summary>
    private static async Task CleanupAsync()
    {
        if (!string.IsNullOrEmpty(_bucketName))
        {
            Console.WriteLine($"Deleting bucket {_bucketName} and its contents...");
            bool result = await _s3Wrapper.DeleteBucketAsync(_bucketName);

            if (result)
            {
                Console.WriteLine("Bucket deleted successfully");
            }
            else
            {
                Console.WriteLine("Failed to delete bucket - it may have been
already deleted");
```

```
            }
        }
    }
}
```

Migrasi proyek Anda untuk AWS SDK untuk .NET

Bagian ini memberikan informasi tentang tugas migrasi untuk versi 4 AWS SDK untuk .NET, dan petunjuk tentang cara melakukan tugas tersebut. Untuk informasi tentang migrasi ke versi sebelumnya (misalnya, migrasi dari V2 ke V3, atau dari satu sub-versi V3 ke versi lainnya), lihat Memigrasi proyek Anda di Panduan AWS SDK untuk .NET Pengembang (versi 3).

Lihat juga Apa yang baru danPlatform yang didukung.

Topik

- Migrasi ke versi 4 AWS SDK untuk .NET
- Migrasi dari .NET Standard 1.3

Migrasi ke versi 4 AWS SDK untuk .NET

AWS SDK untuk .NET Versi 4 (V4) memiliki sejumlah besar perubahan yang melanggar dari versi 3 (V3) SDK. Topik ini menjelaskan perubahan yang melanggar di versi 4 dan kemungkinan pekerjaan yang mungkin perlu Anda lakukan untuk memigrasikan lingkungan atau kode Anda dari V3. Untuk informasi tambahan tentang perubahan penting lainnya dalam SDK, lihat sumber daya berikut:

- Masalah pelacak pengembangan pada:. GitHub https://github.com/aws/aws-sdk-net/issues/3362
- Posting blog Pratinjau 1 dari AWS SDK untuk .NET V4.
- Posting blog Pratinjau 4 dari AWS SDK untuk .NET V4.
- Posting blog Ketersediaan Umum AWS SDK untuk .NET V4.0.

.NET Framework

Target.NET Framework 3.5 telah dihapus dari V4. AWS SDK untuk .NET Akibatnya, SDK tidak lagi mendukung .NET Framework 3.5. Versi SDK ini dikompilasi terhadap .NET Framework 4.7.2 dan berjalan di runtime .NET 4.0. Untuk informasi selengkapnya, lihat Platform yang didukung.

Tipe nilai

Properti yang menggunakan tipe nilai di kelas yang digunakan untuk membuat permintaan dan tanggapan telah diubah untuk menggunakan tipe nilai nullable. Properti dengan jenis berikut telah diubah:

- `bool`telah diubah menjadi `bool?`
- `double`telah diubah menjadi `double?`
- `int`telah diubah menjadi `int?`
- `float`telah diubah menjadi `float?`
- `long`telah diubah menjadi `long?`
- `Datetime`telah diubah menjadi `Datetime?`

Untuk informasi tambahan tentang perubahan ini, lihat posting blog Pratinjau 1 dari AWS SDK untuk .NET V4.

Koleksi

Properti yang menggunakan koleksi di kelas yang digunakan untuk membuat permintaan dan tanggapan sekarang default untuk`null`. Akibatnya, kode Anda perlu memverifikasi bahwa koleksi tidak nol sebelum mencoba menggunakannya. Contoh:

```
var sqsClient = new AmazonSQSClient();
var listResponse = await sqsClient.ListQueuesAsync(new ListQueuesRequest());
if (listResponse.QueueUrls != null)
{
    foreach (string qUrl in listResponse.QueueUrls)
    {
        // Perform operations on each queue such as displaying all the attributes.
    }
}
```

Perilaku V3 untuk menginisialisasi koleksi dapat dipulihkan dengan menyetel `Amazon.AWSConfigs.InitializeCollections` ke. `true` Properti ini juga ada di V3 untuk pengguna yang ingin mencoba perubahan perilaku ini sebelum memutakhirkan ke V4.

Untuk informasi tambahan tentang perubahan ini, lihat posting blog Pratinjau 1 dari AWS SDK untuk .NET V4.

AWS Security Token Service (STS)

- Titik akhir regional

Saat menggunakan penyedia kredensi yang mengandalkan AWS STS, panggilan selalu menggunakan titik akhir regional. Ini berbeda dari V3 SDK, yang menggunakan `us-east-1` Wilayah secara default saat berjalan di partisi publik terlepas dari Wilayah yang dikonfigurasi.

- **`StsRegionalEndpointsValue`**Enum

  `StsRegionalEndpointsValue`Enum telah dihapus dari namespace Amazon.Runtime. Kode apa pun yang menggunakan enum itu harus dihapus.

- **`STSAssumeRoleAWSCredentials`**Kelas

  Penyedia kredensi peran STS yang tidak digunakan lagi,`STSAssumeRoleAWSCredentials`, telah dihapus dari Amazon. SecurityTokennamespace. Gunakan AssumeRoleAWSCredentialsdari Amazon.Runtime sebagai gantinya.

Perubahan yang berkaitan dengan **`ClientConfig`**

Amazon.Runtime. ClientConfigclass adalah kelas dasar dari kelas konfigurasi klien layanan seperti Amazons3config. Perubahan berikut dibuat untuk kelas dasar ini.

- Mode coba lagi default

  `RetryMode`Properti defaultnya bukan. `Standard` `Legacy` Akibatnya, `Legacy` nilai telah dihapus dari Amazon.Runtime. RequestRetryModeenum.

- Mode konfigurasi default

  `DefaultConfigurationMode`Properti defaultnya bukan. `Standard` `Legacy` Akibatnya, `Legacy` nilai telah dihapus dari Amazon.Runtime. DefaultConfigurationModeenum.

- **`ReadWriteTimeout`**Properti

  `ReadWriteTimeout`Properti usang telah dihapus dari semua target kecuali .NET Framework 4.7.2.

AWSSDK.Extensions. NETCore.Setup paket NuGet

AWSSDK.Extensions. NETCore NuGet Paket .Setup telah diperbarui untuk mengatasi masalah yang ada di V3 SDK serta untuk membuat paket aman untuk AOT Asli. Perubahan ini dirangkum di bawah ini. Untuk informasi rinci lihat PR 3353 di aws-sdk-netrepositori di. GitHub

- **`DefaultClientConfig`**Kelas

  `DefaultClientConfig`Kelas tidak lagi diwarisi dari kelas dasar konfigurasi klien layanan Amazon.Runtime. ClientConfig. Properti yang relevan dari `ClientConfig` telah direplikasi `DefaultClientConfig` menggunakan tipe nilai nullable. Perubahan ini memungkinkan kita untuk mendeteksi kapan nilai telah ditetapkan `DefaultClientConfig` saat menyalin nilai ke konfigurasi yang dibuat untuk klien layanan.

  Salah satu hasil khusus dari perubahan ini `DefaultClientConfig.HttpClientFactory` adalah bahwa tidak lagi tersedia di V4. Gunakan `AWSConfigs.HttpClientFactory` sebagai gantinya. Untuk informasi tambahan, lihat GitHub edisi 3790.

  Hasil lain dari perubahan ini adalah bahwa kelebihan generik dari metode `IConfiguration.GetAWSOptions` ekstensi yang menerima objek konfigurasi layanan telah dihapus. Kelebihan non-generik harus digunakan sebagai gantinya, dan SDK akan secara otomatis menangani pengaturan khusus layanan yang mengisi. Untuk informasi tambahan, lihat GitHub edisi 3866.
- AOT asli

  Mekanisme baru untuk membuat klien layanan yang menggunakan metode antarmuka statis C # 11 telah ditambahkan ke paket. Perubahan ini menghilangkan kebutuhan untuk melakukan beban tipe Assembly untuk membuat instance klien layanan, termasuk manipulasi string dari nama antarmuka layanan untuk menghitung jenis klien layanan, yang tidak kompatibel dengan AOT Asli. Perubahan ini hanya tersedia untuk.NET 8 dan yang lebih baru; versi yang lebih lama masih menggunakan mekanisme asli.

Untuk informasi tambahan dalam panduan ini tentang paket ini, lihatAWSSDK.Ekstensi. NETCore.Setup dan IConfiguration. Kode sumber untuk paket ini aktif GitHub dihttps://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.NETCore.Setup.

`CookieSigner` dan **`UrlSigner`**

`UrlSigner`Ekstensi `CookieSigner` dan untuk Amazon CloudFront telah dipindahkan ke paket ekstensi terpisah yang disebut AWSSDK.Extensions. CloudFront.Penandatangan. Perubahan ini untuk mendukung OpenSSL 3 dan mengambil ketergantungan pada .Cryptography. BouncyCastle

Kode sumber untuk paket ini aktif GitHub dihttps://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.CloudFront.Signers.

DateTime versus UTC DateTime

Beberapa kelas V3 memiliki DateTime properti yang ditandai sebagai “usang” atau “usang”, serta properti UTC alternatif. DateTime Di kelas ini, DateTime properti usang telah dihapus, dan nama DateTime properti UTC telah diubah menjadi nama asli properti. DateTime

Berikut ini adalah beberapa contoh kelas yang perubahan ini telah diterapkan.

- DescribeSpotPriceHistoryRequest:
  - `StartTime`Properti usang telah dihapus, dan nama `StartTimeUtc` properti telah diubah menjadi ""StartTime.
  - `EndTime`Properti usang telah dihapus, dan nama `EndTimeUtc` properti telah diubah menjadi ""EndTime.
- CreateFleetRequest
  - `ValidFrom`Properti usang telah dihapus, dan nama `ValidFromUtc` properti telah diubah menjadi ""ValidFrom.
  - `ValidUntil`Properti usang telah dihapus, dan nama `ValidUntilUtc` properti telah diubah menjadi ""ValidUntil.

Perubahan ini dapat menyebabkan waktu offset jika aplikasi menggunakan properti asli yang usang DateTime. Kesalahan waktu kompilasi akan terjadi untuk kode yang menggunakan properti UTC DateTime .

DateTime penguraian

DateTimeUnmarshaller Kelas telah diperbarui. Kelas ini telah mengurai dan mengembalikan DateTime string sebagai waktu setempat. Dalam beberapa kasus, nilai-nilai ini dikonversi kembali ke UTC karena pembaruan sebelumnya, tetapi tidak selalu. Sekarang, DateTime string yang tidak diatur diasumsikan sebagai UTC dan akan ditentukan dan tidak diatur sebagai UTC. Pembaruan ini mencakup perubahan perilaku berikut.

Properti stempel waktu tertentu yang didasarkan pada DateTime kelas sedang diuraikan ke dalam waktu lokal. Ini termasuk respon unmarshallers untuk stempel waktu dan daftar stempel waktu untuk format dan. `TimestampFormat.ISO8601` `TimestampFormat.RFC822` DateTime penguraian telah diperbarui untuk mengembalikan waktu UTC sebagai gantinya.

`ConvertFromUnixEpochSeconds` dan `ConvertFromUnixEpochMilliseconds`

ConvertFromUnixEpochMilliseconds Metode ConvertFromUnixEpochSeconds dan, yang mengubah detik epoch Unix menjadi DateTime struktur, mengembalikan waktu Unix Epoch sebagai waktu lokal, bukan waktu UTC. Metode ini sekarang mengembalikan waktu UTC.

Pencatatan log

Cara Anda mengaktifkan login di SDK telah diperbarui untuk V4. Logging ke konsol dan diagnostik sistem bekerja sama dengan V3; yaitu, dengan mengatur `LoggingConfig.LogTo` properti AWSConfigskelas ke salah satu atau`LoggingOptions.Console`. `LoggingOptions.SystemDiagnostics` LoggingOptionsOpsi untuk `log4net` telah dihapus bersama dengan logika internal SDK untuk menggunakan refleksi untuk dilampirkan ke instance dalam memori. `log4net`

Untuk menyertakan log SDK ke kerangka logging, paket adaptor terpisah digunakan untuk menghubungkan SDK dengan kerangka logging. Gunakan AWSSDKpaket.Extensions.Logging.Logging.Log4 untuk dan NetAdaptor .Extensions.Logging. `log4net` AWSSDK ILoggerPaket adaptor untuk`Microsoft.Extensions.Logging`. Contoh kode berikut menunjukkan cara mengonfigurasi logging dalam dua kasus ini.

Contoh konfigurasi untuk log4net

Tambahkan `AWSSDK.Extensions.Logging.Log4NetAdaptor` NuGet paket dan panggil `ConfigureAWSSDKLogging` metode statis dari`Log4NetAWSExtensions`.

```
using Amazon.DynamoDBv2;
using Amazon.Extensions.Logging.Log4NetAdaptor;
using log4net;

[assembly: log4net.Config.XmlConfigurator(ConfigFile = "log4net.config")]

Log4NetAWSExtensions.ConfigureAWSSDKLogging();
var logger = LogManager.GetLogger(typeof(Program));
```

Contoh konfigurasi untuk Microsoft.Extensions.Logging

Tambahkan `AWSSDK.Extensions.Logging.ILoggerAdaptor` NuGet paket dan panggil metode `ConfigureAWSSDKLogging` ekstensi dari `ILoggerFactory` antarmuka.

```
var builder = WebApplication.CreateBuilder(args);

var app = builder.Build();

app.Services.GetRequiredService<ILoggerFactory>()
    .ConfigureAWSSDKLogging();
```

Support untuk HTTP 2

Support untuk HTTP 2 telah ditambahkan untuk mengaktifkan bi-directional streaming. Untuk mengetahui informasi selengkapnya, lihat Support untuk HTTP 2.

Sign-on tunggal

Nilai default `SupportsGettingNewToken` properti kelas SSOAWSCredentialsOptions telah diubah dari `true` kefalse. Jika Anda memiliki aplikasi yang menggunakan SSOAWSCredentialskelas untuk mendapatkan kredensi SSO, Anda mungkin perlu mengatur properti ke. `Options.SupportsGettingNewToken` `true` Untuk contoh konfigurasi ini, lihat contoh kode diTutorial untuk SSO hanya menggunakan aplikasi.NET. Untuk informasi tambahan, lihat PR 3737 di aws-sdk-net GitHub repositori.

Perubahan khusus untuk DynamoDB

Perubahan berikut khusus untuk Amazon DynamoDB. Banyak dari mereka melanggar perubahan.

Untuk informasi tambahan tentang perubahan DynamoDB di V4 dari, lihat posting blog AWS SDK untuk .NET Pratinjau 4 dari V4. AWS SDK untuk .NET

Buka untuk melihat item

Perubahan V4 dalam SDK untuk DynamoDB mengatasi beberapa masalah seputar stabilitas pengujian, tetapi terutama berpusat di sekitar pustaka tingkat tinggi:

- Model Dokumen .NET, dinamai DocumentModeldalam kode.
- .NET Object Persistence Model, dinamai DataModeldalam kode.

Untuk informasi rinci tentang mode pemrograman ini, lihat DynamoDB di panduan ini.

Model Dokumen: Pengecualian yang diperbarui untuk antarmuka yang diejek **`IAmazonDynamoDB`**

Dalam model dokumen sebelum V4 SDK, jika Tabel diinisialisasi dengan antarmuka DynamoDB IAmazon tiruan, itu akan kembali. `NullReferenceException` V4 dari SDK kembali `InvalidOperationException` sebagai gantinya. `Table`Metode asinkron harus bekerja dengan klien tiruan tetapi Anda mungkin masih melihat pengecualian saat memanggil metode sinkron dari. `.NET/Core/Standard`

Untuk informasi lebih lanjut tentang perubahan ini, lihat PR 3388 di. GitHub

Model Dokumen: **`FromJson`** dan **`ToJson`** metode

`ToJson`Metode `FromJson` dan kelas Document sekarang digunakan `System.Text.Json` bukan LitJson untuk serialisasi, dan LitJson telah dihapus dari V4 SDK. Manfaat menggunakan `System.Text.Json` adalah bahwa parser ini mendukung penggunaan `Decimal` tipe.NET, yang mendukung presisi yang lebih tinggi untuk properti floating point numerik.

Model Persistensi Objek: Kelas **`DynamoDBOperationConfig`**

Dalam model persistensi objek, perubahan berikut telah dilakukan pada kelas Dynamo Config DBOperation bersama:

- Kelas telah dipisahkan menjadi kelas-kelas operasi-spesifik baru seperti SaveConfig, LoadConfig, dan. QueryConfig Metode yang diambil `DynamoDBOperationConfig` telah ditandai sebagai usang dan dapat dihapus di masa depan.

  Untuk informasi lebih lanjut tentang perubahan ini, lihat PR 3421 di GitHub.

- `DisableFetchingTableMetadata`Properti `MetadataCachingMode` dan telah dihapus dari kelas. Properti ini tidak termasuk dalam kelas khusus operasi baru yang disebutkan sebelumnya. Properti yang dihapus adalah pengaturan tingkat tabel yang harus ditentukan pada `Context` properti global kelas DynamoDB atau pada kelas AWSConfigsDynamo Config. DBContext

  Untuk informasi lebih lanjut tentang perubahan ini, lihat PR 3422 di GitHub.

- Kelas tidak lagi mewarisi dari kelas Dynamo Config DBContext. Ini mencegah Anda meneruskan `DynamoDBOperationConfig` objek ke konstruktor untuk Dynamo DBContext, di mana beberapa properti pada konfigurasi khusus operasi (seperti`OverrideTableName`) tidak berlaku.

  Untuk informasi lebih lanjut tentang perubahan ini, lihat PR 3422 di GitHub.

Model Persistensi Objek: Polimorfisme

DBPolymorphicTypeAttributeKelas Dynamo ditambahkan ke model ketekunan objek. Kelas ini memungkinkan dukungan untuk serialisasi dan deserialisasi tipe polimorfik. Untuk informasi lebih lanjut, lihat PR 3643 di GitHub.

Model Dokumen dan Model Persistensi Objek: Operasi Mockable

Antarmuka khusus operasi baru telah ditambahkan yang memungkinkan pelanggan untuk mengejek operasi DynamoDB. Metode pabrik pada IDynamoDBContextantarmuka telah diperbarui untuk mengembalikan antarmuka baru.

Untuk informasi lebih lanjut tentang perubahan ini, lihat PR 3450 di GitHub.

- Model Kegigihan Objek
  - `BatchGet`Operasi tiruan melalui `IBatchGet` dan `IMultiTableBatchGet` antarmuka.
  - `BatchWrite`Operasi tiruan melalui `IBatchWrite` dan `IMultiTableBatchWrite` antarmuka.
  - `TransactGet`Operasi tiruan melalui `ITransactGet` dan `IMultiTableTransactGet` antarmuka.
  - `TransactWrite`Operasi tiruan melalui `ITransactWrite` dan `IMultiTableTransactWrite` antarmuka.
  - Mock `Scan` dan `Query` operasi melalui `IAsyncSearch` antarmuka.
- Model Dokumen
  - `Table`Operasi tiruan melalui `ITable` antarmuka.
  - Mock `Scan` dan `Query` operasi melalui `ISearch` antarmuka.
  - `TransactWrite`Operasi tiruan melalui `IDocumentTransactWrite` dan `IMultiTableDocumentTransactWrite` antarmuka.
  - `TransactGet`Operasi tiruan melalui `IDocumentTransactGet` dan `IMultiTableDocumentTransactGet` antarmuka.
  - `BatchWrite`Operasi tiruan melalui `IDocumentBatchWrite` dan `IMultiTableDocumentBatchWrite` antarmuka.
  - `BatchGet`Operasi tiruan melalui `IDocumentBatchGet` dan `IMultiTableDocumentBatchGet` antarmuka.

Model Dokumen dan Model Persistensi Objek: Dukungan untuk AOT Asli

Batasan Native AOT adalah dukungan untuk tipe.NET bersarang. Dalam beberapa kasus, tipe bersarang ini mungkin luput dari perhatian oleh komponen pemangkasan dari kompiler.NET. Dalam hal ini, Anda mungkin menerima pengecualian seperti: "`System.InvalidOperationException: Type <type> is unsupported, it cannot be instantiated.`"

Anda dapat mengatasi batasan ini dengan menambahkan `DynamicDependency` suatu tempat di jalur kode yang memberi tahu pemangkas tentang ketergantungan pada sub-tipe. Konstruktor tipe.NET tingkat atas yang disimpan adalah tempat yang mungkin. Contoh kode berikut menunjukkan cara menggunakan `DynamicDependency` atribut:

```
[DynamoDBTable("TestTable")]
class TypeWithNestedTypeProperty
{
    [DynamicDependency(DynamicallyAccessedMemberTypes.All, typeof(SubType))]
    public TypeWithNestedTypeProperty()
    {

    }

    [DynamoDBHashKey]
    public string Id { get; set; }
    public string Name { get; set; }

    public SubType SubType { get; set; }
}

class SubType
{
    public string SubName { get; set; }
}
```

Dinamo DBStreams

Dynamo DBStreams telah dihapus dari AWSSDKpaket.DynamoDB dan tersedia dalam NuGet paketnya sendiri, AWSSDK.Dynamo, dan memiliki namespace sendiriDBStreams,. `Amazon.DynamoDBStreams`

Izinkan penghapusan **`TableNamePrefix`** nilai

Anda sekarang dapat menghapus nilai `TableNamePrefix` properti di kelas Dynamo DBContext Config pada tingkat operasi individu. Untuk informasi lebih lanjut tentang perubahan ini, lihat PR 3476 di GitHub.

`RetrieveDateTimeInUtc`properti

Untuk kelas Dynamo DBContext Config, nilai default properti telah diubah menjadi. `RetrieveDateTimeInUtc` `true`

`DynamoDBContextTableNamePrefix`properti

Dihapus `DynamoDBContextTableNamePrefix` properti dari kelas AWSConfigsDynamoDB. Pengguna harus menelepon `AWSConfigsDynamoDB.Context.TableNamePrefix` alih-alih

Perubahan khusus untuk EC2

Perubahan berikut khusus untuk Amazon EC2. Sebagian besar atau semuanya melanggar perubahan.

Buka untuk melihat item

`GetDecryptedPassword`

`GetDecryptedPassword`Ekstensi untuk Amazon EC2 telah dipindahkan ke paket ekstensi terpisah yang disebut AWSSDK .Extensions.EC2. DecryptPassword. Perubahan ini untuk mendukung OpenSSL 3 dan mengambil ketergantungan pada .Cryptography. BouncyCastle

Kode sumber untuk paket ini aktif GitHub dihttps://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.EC2.DecryptPassword.

Dukungan untuk Amazon EC2 IMDSv1

Support for Instance Metadata Service Version 1 (IMDSv1) telah dihapus. V4 SDK selalu menggunakan Instance Metadata Service Version 2 (IMDSv2) saat mengambil kredensyal dan metadata lainnya dari IMDS. Untuk informasi selengkapnya tentang IMDS, lihat Menggunakan IMDS di Panduan Pengguna Amazon EC2.

Elemen pemrograman yang diubah atau dihapus

- Seluruh `Amazon.EC2.Import` namespace dan kode telah dihapus.

- Seluruh `Amazon.EC2.Util` namespace dan kode telah dihapus, yang mencakup utilitas AMI yang digunakan untuk mencari EC2 AMIs untuk Windows.
- `IpRanges`Properti usang telah dihapus dari kelas. IpPermission Gunakan `Ipv6Ranges` properti `Ipv4Ranges` atau sebagai gantinya.
- Bidang usang berikut telah dihapus dari EC2InstanceMetadatakelas:`EC2_METADATA_SVC`,,, `EC2_METADATA_ROOT` `EC2_USERDATA_ROOT``EC2_DYNAMICDATA_ROOT`, dan. `EC2_APITOKEN_URL`

Perubahan khusus untuk S3

Perubahan berikut khusus untuk Amazon S3. Sebagian besar atau semuanya melanggar perubahan.

Buka untuk melihat item

Wilayah AWS us-east-1

Klien layanan Amazon S3 yang dikonfigurasi untuk `us-east-1` Wilayah tidak dapat lagi mengakses bucket di Wilayah lain. Bucket harus diakses dengan klien layanan S3 yang dikonfigurasi untuk Wilayah tempat bucket berada.

Untuk informasi tambahan tentang perubahan ini, lihat posting blog Pratinjau 4 dari AWS SDK untuk .NET V4.

Klien enkripsi S3

Klien enkripsi Amazon S3, yang didefinisikan dalam `Amazon.S3.Encryption` namespace, telah dihapus dari paket.S3. AWSSDK Klien ini telah dipindahkan ke paketnya sendiri yang disebut Amazon.Extensions.S3.Encryption, dan dokumentasinya ada di. https://aws.github.io/amazon-s3-encryption-client-dotnet/api/Amazon.Extensions.S3.Encryption.html Untuk informasi tentang migrasi, lihatthe section called “Migrasi Klien Enkripsi S3 (V1 ke V2)”. Untuk informasi selengkapnya tentang enkripsi S3, lihat Algoritma enkripsi yang didukung di Panduan Pengembang Klien Enkripsi Amazon S3.

Arahan penandaan S3 untuk **`CopyObject`**

`TaggingDirective`Properti telah diekspos sebagai milik umum CopyObjectRequestkelas, yang digunakan oleh `AmazonS3Client.CopyObject` metode. Properti ini sesuai dengan `x-amz-tagging-directive` parameter Amazon S3, seperti yang didefinisikan dalam tindakan. CopyObject

Arahan penandaan tidak lagi secara otomatis diatur ke COPY. Jika pengembang tidak menentukan arahan penandaan, backend S3 secara otomatis mengasumsikan itu adalah COPY, tetapi jika pengembang secara eksplisit menyetel properti ke null, maka nilainya tidak disetel sama sekali.

`UseArnRegion`Properti untuk konfigurasi S3

`UseArnRegion`Properti kelas Amazon.s3.amazons3config telah diperbarui sedemikian rupa sehingga variabel `AWS_S3_USE_ARN_REGION` lingkungan lebih diutamakan daripada pengaturan dalam file bersama. `s3_use_arn_region` AWS `config` Untuk informasi selengkapnya tentang variabel dan pengaturan ini, lihat Referensi pengaturan di Panduan Referensi Alat AWS SDKs dan Alat.

garis miring terkemuka untuk metode **`CopyObject`** dan **`CopyPart`**

garis miring terkemuka tidak akan lagi dipangkas untuk Amazon `CopyObject` `CopyPart` S3 dan metode. `DisableTrimmingLeadingSlash`Properti telah dihapus dari CopyPartRequestkelas CopyObjectRequestdan.

`DoesS3BucketExist...`Metodenya

Usang `DoesS3BucketExist` dan `DoesS3BucketExistAsync` metode telah dihapus dari kelas Amazons3Util, yang mengimplementasikan antarmuka AmazonS3. ICore Metode ini dihapus karena mereka selalu menggunakan HTTP. Gunakan Doess3 BucketExist V2 dan BucketExistDoess3 V2Async sebagai gantinya.

SDK selalu menggunakan SiGv4

Versi 4 AWS SDK untuk .NET selalu menggunakan AWS Signature Version 4 (SiGv4) untuk menandatangani permintaan. Perubahan ini menghasilkan perubahan terkait berikut:

- `UseSignatureVersion4`Properti kelas AWSConfigsS3 telah dihapus.
- `SignatureVersion`Properti Amazon.Runtime. ClientConfigkelas telah dihapus. Properti ini hanya digunakan oleh Amazon S3 untuk kompatibilitas mundur.
- `RegionEndpoint.Endpoint`Kelas telah dihapus. Ini termasuk `SignatureVersionOverride` properti, yang digunakan untuk mengganti versi tanda tangan untuk Amazon S3. Gunakan `client.DetermineServiceOperationEndPoint()` metode khusus layanan sebagai gantinya.
- Metode yang diperbarui Amazons3util. PostUploaddan S3PostUploadSignedPolicy. GetSignedPolicyuntuk menggunakan SiGv4. Akibatnya,

`S3PostUploadSignedPolicy.GetSignedPolicyV4` metode telah dihapus karena `GetSignedPolicy` sekarang melakukan fungsi yang sama. Selain itu, `GetSignedPolicy` telah diberikan parameter ketiga untuk titik akhir Wilayah.

`PutACL`Metode `GetACL` dan

`PutACL`Metode `GetACL` dan dari kelas Amazons3Client telah ditandai sebagai usang. Untuk mengakses fungsionalitas metode ini, gunakan metode baru berikut sebagai gantinya:`GetBucketACL`,`PutBucketACL`,`GetObjectACL`, dan`PutObjectACL`.

Elemen pemrograman usang dihapus

Sejumlah elemen pemrograman implementasi Amazon S3 telah dihapus dari V4 SDK, termasuk nilai enumerasi, jenis, metode, ruang nama, dll. Ini tercantum di bawah ini, jika belum dibahas sebelumnya, bersama dengan langkah-langkah potensial yang dapat Anda ambil untuk mengakomodasi penghapusan mereka.

- `DisableMD5Stream`Properti telah dihapus dari TransferUtilityUploadRequestkelas. Gunakan `DisableDefaultChecksumValidation` properti sebagai gantinya.

  Selain itu, `CalculateContentMD5Header` properti telah dihapus dari `TransferUtilityUploadRequest` kelas. Properti ini tidak lagi diperlukan karena SDK menghitung checksum secara default.
- `ServerSideEncryptionKeyManagementServiceKeyId`Properti `ServerSideEncryptionMethod` dan telah dihapus dari CopyPartRequestkelas. Gunakan properti dengan nama yang sama di InitiateMultipartUploadRequestkelas sebagai gantinya, yang digunakan dalam beberapa `InitiateMultipartUpload...` metode kelas Amazons3Client.
- `Expires`Properti telah dihapus dari GetObjectResponsekelas. Gunakan `ExpiresString` properti sebagai gantinya. String mungkin tidak dalam format stempel waktu yang valid, jadi kode Anda harus menggunakan `TryParse` metode saat mengonversi ke a. `DateTime`
- Wilayah AWS Pengidentifikasi usang telah dihapus dari enumerasi S3Region.
- `Prefix`Properti telah dihapus dari LifecycleRulekelas. Gunakan `Filter` properti sebagai gantinya.

  Selain itu, `Transition` properti `NoncurrentVersionTransition` dan telah dihapus dari `LifecycleRule` kelas. Gunakan `NoncurrentVersionTransitions` dan `Transitions` koleksi sebagai gantinya.

- `Event`Properti telah dihapus dari TopicConfigurationkelas. Gunakan `Events` koleksi sebagai gantinya.
- CalculateContentMD5Properti header. Properti ini tidak perlu lagi disetel karena SDK akan menghitung checksum secara default.
- `Bucket`Properti telah dihapus dari SelectObjectContentRequestkelas. Gunakan `BucketName` properti sebagai gantinya.
- `NumberOfUploadThreads`Properti telah dihapus dari TransferUtilityConfigkelas. Gunakan `ConcurrentServiceRequests` properti sebagai gantinya.

Elemen pemrograman yang telah dihapus

Sejumlah elemen pemrograman telah dihapus dari V4 SDK, termasuk nilai enumerasi, jenis, metode, ruang nama, dll. Ini tercantum di bawah ini, jika belum dibahas sebelumnya, bersama dengan langkah-langkah potensial yang dapat Anda ambil untuk mengakomodasi penghapusan mereka.

Buka untuk melihat item

Namespace **`Amazon.Auth.AccessControlPolicy.ActionIdentifiers`**

`Amazon.Auth.AccessControlPolicy.ActionIdentifiers`Namespace telah dihapus. Ini termasuk pengidentifikasi tindakan IAM, yang didefinisikan di kelas. `IdentityandAccessManagementActionIdentifiers` Kode yang menggunakan pengidentifikasi tindakan ini harus diubah untuk menggunakan nilai string dari nama tindakan.

Untuk informasi tambahan, lihat the section called “Membuat kebijakan terkelola dari JSON” dan Ringkasan kebijakan JSON di Panduan Pengguna IAM.

`ClientConfig`Kelas

Amazon.Runtime. ClientConfigclass adalah kelas dasar dari kelas konfigurasi klien layanan seperti Amazons3config. Elemen pemrograman berikut telah dihapus dari kelas ini.

- `DetermineDnsSuffix`Metode `DetermineServiceURL` dan telah dihapus. Gunakan `DetermineServiceOperationEndpoint` metode klien layanan sebagai gantinya; misalnya, Amazons3Client. DetermineServiceOperationEndpoint.
- `ReadEntireResponse`Properti telah dihapus. Gunakan salah satu dari berikut ini sebagai gantinya:
  - `LogResponses`Properti dari AWSConfigs. LoggingConfigkelas.

- `LogResponse`Properti konfigurasi klien; misalnya, Amazons3config.

Namespace **`Amazon.Runtime`**

Namespace Amazon.Runtime telah diperbarui sebagai berikut:

- `ECSTaskCredentials`Kelas usang telah dihapus dari namespace. Gunakan GenericContainerCredentialspenyedia sebagai gantinya, yang juga mendukung Amazon EKS Pod Identities.
- Usang `StoredProfileAWSCredentials` dan `StoredProfileCredentials` kelas telah dihapus dari namespace. Gunakan SDKCredentialsFile Net atau SharedCredentialsFilekelas Amazon.Runtime. CredentialManagementnamespace sebagai gantinya.
- `HasCachedAccessTokenAvailable`Metode usang SSOAWSCredentialskelas telah dihapus dari namespace.
- `EnvironmentAWSCredentials`Kelas usang telah dihapus dari namespace. Gunakan AppConfigAWSCredentialskelas sebagai gantinya.
- `StoredProfileFederatedCredentials`Kelas usang telah dihapus dari namespace. Gunakan AWSCredentials kelas Federasi sebagai gantinya.
- Kelas usang berikut telah dihapus dari namespace:`EnvironmentVariableAWSEndpointDiscoveryEnabled`,, dan. `ProfileAWSEndpointDiscoveryEnabled` `FallbackEndpointDiscoveryEnabledFactory`
- `UseSigV4`Properti usang telah dihapus dari kelas. AmazonWebServiceRequest Gunakan `SignatureVersion` properti sebagai gantinya.
- `ProfileIniFile`Kelas di `Amazon.Runtime.Internal.Util` namespace memiliki metode kelebihan beban yang disebut. `TryGetSection` Versi metode yang tidak mendukung `out` parameter untuk `nestedProperties` telah dihapus dari kelas.
- `EventBridgeSigner`Kelas usang di `Amazon.Runtime.Internal.Auth` namespace telah dihapus.
- Properti `Parameters` kamus usang telah dihapus dari kelas. WebServiceRequestEventArgs Gunakan `ParameteCollection` properti sebagai gantinya.

BouncyCastle

Salinan sumber BouncyCastle telah dihapus dari V4 SDK.

`StoredProfileSAMLCredentials`Kelas

`StoredProfileSAMLCredentials`Kelas usang di Amazon. SecurityTokenNamespace .SALL telah dihapus. Gunakan AWSCredentials kelas Federasi di namespace Amazon.Runtime sebagai gantinya.

`AWSSDKUtils`Kelas

Metode berikut telah dihapus dari AWSSDKUtilskelas:`ResolveResourcePath`,`ProtectEncodedSlashUrlEncode`, dan`ConvertToUnixEpochMilliSeconds`.

`ProfileManager`Kelas

`ProfileManager`Kelas usang telah dihapus dari namespace Amazon.Util. Gunakan SDKCredentialsFile Net atau SharedCredentialsFilekelas dari Amazon.Runtime. CredentialManagementnamespace sebagai gantinya.

`AWSConfigs`Kelas

Properti usang berikut telah dihapus dari AWSConfigskelas:`Logging`,`ResponseLogging`, dan. `LogMetrics` Gunakan `LoggingConfig` properti sebagai gantinya.

`ConditionFactory`Kelas

Metode dengan tanda tangan berikut telah dihapus dari ConditionFactorykelas:`NewCondition(ConditionFactory.DateComparisonType, DateTime)`. Gunakan NewConditionUtcmetode sebagai gantinya.

CloudFront Utilitas Amazon

`Amazon.CloudFront.Util`Namespace dan `AmazonCloudFrontUtil` kelas usang telah dihapus.

AWS IoT

Di ListPrincipalThingsResponsekelas, kustomisasi lama untuk `NextToken` penggantian telah dihapus demi pagination.

AWS Lambda

`Invoke...`Metode AmazonLambdaClientkelas berikut telah dihapus karena nama-nama yang membingungkan.

- Metode V3 dengan tanda tangan berikut telah dihapus:`InvokeAsyncResponse InvokeAsync(InvokeAsyncRequest)`. Ini adalah metode sinkron di V3 SDK. Gunakan `InvokeResponse Invoke(InvokeRequest)` (untuk pemrosesan sinkron) atau `Task InvokeAsync(InvokeRequest, CancellationToken)` (untuk pemrosesan asinkron) sebagai gantinya.
- Metode V3 dengan tanda tangan berikut telah dihapus:`Task InvokeAsyncAsync(InvokeAsyncRequest, CancellationToken)`. Ini adalah metode asinkron di V3 SDK. Gunakan `Task InvokeAsync(InvokeRequest, CancellationToken)` sebagai gantinya.

SageMaker Runtime Amazon

Konstruktor usang untuk PayloadPartkelas telah dihapus.

Migrasi dari .NET Standard 1.3

Pada 27 Juni 2019 Microsoft mengakhiri dukungan untuk versi .NET Core 1.0 dan .NET Core 1.1. Setelah pengumuman ini, AWS mengakhiri dukungan untuk .NET Standard 1.3 AWS SDK untuk .NET pada tanggal 31 Desember 2020.

AWS terus memberikan pembaruan layanan dan perbaikan keamanan pada AWS SDK untuk .NET penargetan .NET Standard 1.3 hingga 1 Oktober 2020. Setelah tanggal tersebut, target.NET Standard 1.3 masuk ke mode Pemeliharaan, yang berarti tidak ada pembaruan baru yang dirilis; AWS menerapkan perbaikan bug kritis dan patch keamanan saja.

Pada tanggal 31 Desember 2020, dukungan untuk .NET Standard 1.3 AWS SDK untuk .NET sampai pada akhir masa pakainya. Setelah tanggal tersebut tidak ada perbaikan bug atau patch keamanan yang diterapkan. Artefak yang dibangun dengan target itu tetap tersedia untuk diunduh NuGet.

Apa yang perlu Anda lakukan

- Jika Anda menjalankan aplikasi menggunakan .NET Framework, Anda tidak terpengaruh.

- Jika Anda menjalankan aplikasi menggunakan .NET Core 2.0 atau lebih tinggi, Anda tidak terpengaruh.

- Jika Anda menjalankan aplikasi menggunakan .NET Core 1.0 atau .NET Core 1.1, migrasi aplikasi Anda ke versi yang lebih baru .NET Core dengan mengikuti petunjuk migrasi Microsoft. Kami merekomendasikan minimal .NET Core 3.1.

- Jika Anda menjalankan aplikasi penting bisnis yang tidak dapat ditingkatkan saat ini, Anda dapat terus menggunakan versi Anda saat ini. AWS SDK untuk .NET

Jika Anda memiliki pertanyaan atau masalah, hubungi AWS Support.

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
- Identity and Access Management
- Validasi Kepatuhan untuk AWS Produk atau Layanan ini
- Ketahanan untuk AWS Produk atau Layanan ini
- Keamanan Infrastruktur untuk AWS Produk atau Layanan ini
- Menegakkan versi TLS minimum di AWS SDK untuk .NET
- Migrasi Klien Enkripsi S3 (V1 ke V2)
- Migrasi Klien Enkripsi S3 (V2 ke V4)

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. Lihat informasi yang lebih lengkap tentang privasi data dalam Pertanyaan Umum Privasi Data. Lihat informasi tentang perlindungan data di Eropa di pos blog Model Tanggung Jawab Bersama dan GDPR AWS di Blog Keamanan AWS .

Untuk tujuan perlindungan data, kami menyarankan Anda melindungi Akun AWS kredensyal 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 autentikasi multi-faktor (MFA) pada setiap akun.
- Gunakan SSL/TLS untuk berkomunikasi dengan AWS sumber daya. Kami mensyaratkan TLS 1.2 dan menganjurkan TLS 1.3.
- Siapkan API dan pencatatan 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 modul kriptografi tervalidasi FIPS 140-3 saat mengakses AWS melalui antarmuka baris perintah atau API, gunakan titik akhir FIPS. Lihat informasi selengkapnya tentang titik akhir FIPS yang tersedia di 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. Saat Anda memberikan URL ke server eksternal, kami sangat menganjurkan supaya Anda tidak menyertakan informasi kredensial di dalam URL untuk memvalidasi permintaan Anda ke server itu.

Identity and Access Management

AWS Identity and Access Management (IAM) adalah Layanan AWS yang membantu administrator mengontrol akses ke AWS sumber daya dengan aman. Administrator IAM mengontrol siapa yang dapat diautentikasi (masuk) dan diberi wewenang (memiliki izin) untuk menggunakan sumber daya. AWS IAM adalah Layanan AWS yang dapat Anda gunakan tanpa biaya tambahan.

Topik

- Audiens
- Mengautentikasi dengan identitas
- Mengelola akses menggunakan kebijakan
- Bagaimana Layanan AWS bekerja dengan IAM
- Memecahkan masalah AWS identitas dan akses

Audiens

Cara Anda menggunakan AWS Identity and Access Management (IAM) berbeda, tergantung pada pekerjaan yang Anda lakukan. AWS

Pengguna layanan — Jika Anda menggunakan Layanan AWS untuk melakukan pekerjaan Anda, maka administrator Anda memberi Anda kredensyal dan izin yang Anda butuhkan. Saat Anda menggunakan lebih banyak AWS fitur untuk melakukan pekerjaan Anda, Anda mungkin memerlukan izin tambahan. Memahami cara akses dikelola dapat membantu Anda meminta izin yang tepat dari administrator Anda. Jika Anda tidak dapat mengakses fitur AWS, lihat Memecahkan masalah AWS identitas dan akses atau panduan pengguna yang Layanan AWS Anda gunakan.

Administrator layanan — Jika Anda bertanggung jawab atas AWS sumber daya di perusahaan Anda, Anda mungkin memiliki akses penuh ke AWS. Tugas Anda adalah menentukan AWS fitur dan sumber daya mana yang harus diakses pengguna layanan Anda. Kemudian, Anda harus mengirimkan permintaan kepada administrator IAM untuk mengubah izin pengguna layanan Anda. Tinjau informasi di halaman ini untuk memahami konsep dasar IAM. Untuk mempelajari lebih lanjut tentang bagaimana perusahaan Anda dapat menggunakan IAM AWS, lihat panduan pengguna yang Layanan AWS Anda gunakan.

Administrator IAM – Jika Anda adalah administrator IAM, Anda sebaiknya mempelajari detail tentang cara menulis kebijakan untuk mengelola akses ke AWS. Untuk melihat contoh kebijakan AWS berbasis identitas yang dapat Anda gunakan di IAM, lihat panduan pengguna yang Anda gunakan. Layanan AWS

Mengautentikasi dengan identitas

Otentikasi adalah cara Anda masuk AWS menggunakan kredensyal identitas Anda. Anda harus diautentikasi sebagai Pengguna root akun AWS, pengguna IAM, atau dengan mengasumsikan peran IAM.

Anda dapat masuk sebagai identitas federasi menggunakan kredensyal dari sumber identitas seperti AWS IAM Identity Center (Pusat Identitas IAM), autentikasi masuk tunggal, atau kredensyal. Google/ Facebook Untuk informasi selengkapnya tentang cara masuk, lihat Cara masuk ke Akun AWS Anda dalam Panduan Pengguna AWS Sign-In .

Untuk akses terprogram, AWS sediakan SDK dan CLI untuk menandatangani permintaan secara kriptografis. Untuk informasi selengkapnya, lihat AWS Signature Version 4 untuk permintaan API dalam Panduan Pengguna IAM.

Akun AWS pengguna root

Saat Anda membuat Akun AWS, Anda mulai dengan satu identitas masuk yang disebut pengguna Akun AWS root yang memiliki akses lengkap ke semua Layanan AWS dan sumber daya. Kami sangat menyarankan agar Anda tidak menggunakan pengguna root untuk tugas sehari-hari. Untuk tugas yang memerlukan kredensial pengguna root, lihat Tugas yang memerlukan kredensial pengguna root dalam Panduan Pengguna IAM.

Identitas terfederasi

Sebagai praktik terbaik, mewajibkan pengguna manusia untuk menggunakan federasi dengan penyedia identitas untuk mengakses Layanan AWS menggunakan kredensyal sementara.

Identitas federasi adalah pengguna dari direktori perusahaan Anda, penyedia identitas web, atau Directory Service yang mengakses Layanan AWS menggunakan kredensyal dari sumber identitas. Identitas terfederasi mengambil peran yang memberikan kredensial sementara.

Untuk manajemen akses terpusat, kami menyarankan AWS IAM Identity Center. Untuk informasi selengkapnya, lihat Apa itu Pusat Identitas IAM? dalam Panduan Pengguna AWS IAM Identity Center .

Pengguna dan grup IAM

Pengguna IAM adalah identitas dengan izin khusus untuk satu orang atau aplikasi. Sebaiknya gunakan kredensial sementara alih-alih pengguna IAM dengan kredensial jangka panjang. Untuk informasi selengkapnya, lihat Mewajibkan pengguna manusia untuk menggunakan federasi dengan penyedia identitas untuk mengakses AWS menggunakan kredensyal sementara di Panduan Pengguna IAM.

Grup IAM menentukan kumpulan pengguna IAM dan mempermudah pengelolaan izin untuk pengguna dalam jumlah besar. Untuk mempelajari selengkapnya, lihat Kasus penggunaan untuk pengguna IAM dalam Panduan Pengguna IAM.

Peran IAM

Peran IAM adalah identitas dengan izin khusus yang menyediakan kredensial sementara. Anda dapat mengambil peran dengan beralih dari pengguna ke peran IAM (konsol) atau dengan memanggil operasi AWS CLI atau AWS API. Untuk informasi selengkapnya, lihat Metode untuk mengambil peran dalam Panduan Pengguna IAM.

Peran IAM berguna untuk akses pengguna terfederasi, izin pengguna IAM sementara, akses lintas akun, akses lintas layanan, dan aplikasi yang berjalan di Amazon EC2. Untuk informasi selengkapnya, lihat Akses sumber daya lintas akun di IAM dalam Panduan Pengguna IAM.

Mengelola akses menggunakan kebijakan

Anda mengontrol akses AWS dengan membuat kebijakan dan melampirkannya ke AWS identitas atau sumber daya. Kebijakan menentukan izin saat dikaitkan dengan identitas atau sumber daya. AWS mengevaluasi kebijakan ini ketika kepala sekolah membuat permintaan. Sebagian besar kebijakan disimpan AWS sebagai dokumen JSON. Untuk informasi selengkapnya tentang dokumen kebijakan JSON, lihat Gambaran umum kebijakan JSON dalam Panduan Pengguna IAM.

Menggunakan kebijakan, administrator menentukan siapa yang memiliki akses ke apa dengan mendefinisikan principal mana yang dapat melakukan tindakan pada sumber daya apa, dan dalam kondisi apa.

Secara default, pengguna dan peran tidak memiliki izin. Administrator IAM membuat kebijakan IAM dan menambahkannya ke peran, yang kemudian dapat diambil oleh pengguna. Kebijakan IAM mendefinisikan izin terlepas dari metode yang Anda gunakan untuk melakukan operasinya.

Kebijakan berbasis identitas

Kebijakan berbasis identitas adalah dokumen kebijakan izin JSON yang Anda lampirkan ke identitas (pengguna, grup, atau peran). Kebijakan ini mengontrol tindakan apa yang bisa dilakukan oleh identitas tersebut, terhadap sumber daya yang mana, dan dalam kondisi apa. Untuk mempelajari cara membuat kebijakan berbasis identitas, lihat Tentukan izin IAM kustom dengan kebijakan yang dikelola pelanggan dalam Panduan Pengguna IAM.

Kebijakan berbasis identitas dapat berupa kebijakan inline (disematkan langsung ke dalam satu identitas) atau kebijakan terkelola (kebijakan mandiri yang dilampirkan pada banyak identitas). Untuk mempelajari cara memilih antara kebijakan terkelola dan kebijakan inline, lihat Pilih antara kebijakan terkelola dan kebijakan inline dalam Panduan Pengguna IAM.

Kebijakan berbasis sumber daya

Kebijakan berbasis sumber daya adalah dokumen kebijakan JSON yang Anda lampirkan ke sumber daya. Contohnya termasuk kebijakan kepercayaan peran IAM dan kebijakan bucket Amazon S3. Dalam layanan yang mendukung kebijakan berbasis sumber daya, administrator layanan dapat menggunakannya untuk mengontrol akses ke sumber daya tertentu. Anda harus menentukan principal dalam kebijakan berbasis sumber daya.

Kebijakan berbasis sumber daya merupakan kebijakan inline yang terletak di layanan tersebut. Anda tidak dapat menggunakan kebijakan AWS terkelola dari IAM dalam kebijakan berbasis sumber daya.

Daftar kontrol akses (ACLs)

Access control lists (ACLs) mengontrol prinsipal mana (anggota akun, pengguna, atau peran) yang memiliki izin untuk mengakses sumber daya. ACLs mirip dengan kebijakan berbasis sumber daya, meskipun mereka tidak menggunakan format dokumen kebijakan JSON.

Amazon S3, AWS WAF, dan Amazon VPC adalah contoh layanan yang mendukung. ACLs Untuk mempelajari selengkapnya ACLs, lihat Ringkasan daftar kontrol akses (ACL) di Panduan Pengembang Layanan Penyimpanan Sederhana Amazon.

Jenis-jenis kebijakan lain

AWS mendukung jenis kebijakan tambahan yang dapat menetapkan izin maksimum yang diberikan oleh jenis kebijakan yang lebih umum:

- Batasan izin – Menetapkan izin maksimum yang dapat diberikan oleh kebijakan berbasis identitas kepada entitas IAM. Untuk informasi selengkapnya, lihat Batasan izin untuk entitas IAM dalam Panduan Pengguna IAM.
- Kebijakan kontrol layanan (SCPs) — Tentukan izin maksimum untuk organisasi atau unit organisasi di AWS Organizations. Untuk informasi selengkapnya, lihat Kebijakan kontrol layanan dalam Panduan Pengguna AWS Organizations .
- Kebijakan kontrol sumber daya (RCPs) — Tetapkan izin maksimum yang tersedia untuk sumber daya di akun Anda. Untuk informasi selengkapnya, lihat Kebijakan kontrol sumber daya (RCPs) di Panduan AWS Organizations Pengguna.
- Kebijakan sesi – Kebijakan lanjutan yang diteruskan sebagai parameter saat membuat sesi sementara untuk peran atau pengguna terfederasi. Untuk informasi selengkapnya, lihat Kebijakan sesi dalam Panduan Pengguna IAM.

Berbagai jenis kebijakan

Ketika beberapa jenis kebijakan berlaku pada suatu permintaan, izin yang dihasilkan lebih rumit untuk dipahami. Untuk mempelajari cara AWS menentukan apakah akan mengizinkan permintaan saat beberapa jenis kebijakan terlibat, lihat Logika evaluasi kebijakan di Panduan Pengguna IAM.

Bagaimana Layanan AWS bekerja dengan IAM

Untuk mendapatkan tampilan tingkat tinggi tentang cara Layanan AWS bekerja dengan sebagian besar fitur IAM, lihat AWS layanan yang bekerja dengan IAM di Panduan Pengguna IAM.

Untuk mempelajari cara menggunakan yang spesifik Layanan AWS dengan IAM, lihat bagian keamanan dari Panduan Pengguna layanan yang relevan.

Memecahkan masalah AWS identitas dan akses

Gunakan informasi berikut untuk membantu Anda mendiagnosis dan memperbaiki masalah umum yang mungkin Anda temui saat bekerja dengan AWS dan IAM.

Topik

- Saya tidak berwenang untuk melakukan tindakan di AWS
- Saya tidak berwenang untuk melakukan iam: PassRole
- Saya ingin mengizinkan orang di luar saya Akun AWS untuk mengakses AWS sumber daya saya

Saya tidak berwenang untuk melakukan tindakan di AWS

Jika Anda menerima pesan kesalahan bahwa Anda tidak memiliki otorisasi untuk melakukan tindakan, kebijakan Anda harus diperbarui agar Anda dapat melakukan tindakan tersebut.

Contoh kesalahan berikut terjadi ketika pengguna IAM `mateojackson` mencoba menggunakan konsol untuk melihat detail tentang suatu sumber daya *`my-example-widget`* rekaan, tetapi tidak memiliki izin `awes:`*`GetWidget`* rekaan.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

Dalam hal ini, kebijakan untuk pengguna `mateojackson` harus diperbarui untuk mengizinkan akses ke sumber daya *`my-example-widget`* dengan menggunakan tindakan `awes:`*`GetWidget`*.

Jika Anda memerlukan bantuan, hubungi AWS administrator Anda. Administrator Anda adalah orang yang memberi Anda kredensial masuk.

Saya tidak berwenang untuk melakukan iam: PassRole

Jika Anda menerima kesalahan yang tidak diizinkan untuk melakukan `iam:PassRole` tindakan, kebijakan Anda harus diperbarui agar Anda dapat meneruskan peran AWS.

Beberapa Layanan AWS memungkinkan Anda untuk meneruskan peran yang ada ke layanan tersebut alih-alih membuat peran layanan baru atau peran terkait layanan. Untuk melakukannya, Anda harus memiliki izin untuk meneruskan peran ke layanan.

Contoh kesalahan berikut terjadi ketika pengguna IAM bernama `marymajor` mencoba menggunakan konsol tersebut untuk melakukan tindakan di AWS. Namun, tindakan tersebut memerlukan layanan untuk mendapatkan izin yang diberikan oleh peran layanan. Mary tidak memiliki izin untuk meneruskan peran tersebut pada layanan.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

Dalam kasus ini, kebijakan Mary harus diperbarui agar dia mendapatkan izin untuk melakukan tindakan `iam:PassRole` tersebut.

Jika Anda memerlukan bantuan, hubungi AWS administrator Anda. Administrator Anda adalah orang yang memberi Anda kredensial masuk.

Saya ingin mengizinkan orang di luar saya Akun AWS untuk mengakses AWS sumber daya saya

Anda dapat membuat peran yang dapat digunakan pengguna di akun lain atau orang-orang di luar organisasi Anda untuk mengakses sumber daya Anda. Anda dapat menentukan siapa saja yang dipercaya untuk mengambil peran tersebut. Untuk layanan yang mendukung kebijakan berbasis sumber daya atau daftar kontrol akses (ACLs), Anda dapat menggunakan kebijakan tersebut untuk memberi orang akses ke sumber daya Anda.

Untuk mempelajari selengkapnya, periksa referensi berikut:

- Untuk mempelajari apakah AWS mendukung fitur-fitur ini, lihatBagaimana Layanan AWS bekerja dengan IAM.
- Untuk mempelajari cara menyediakan akses ke sumber daya Anda di seluruh sumber daya Akun AWS yang Anda miliki, lihat Menyediakan akses ke pengguna IAM di pengguna lain Akun AWS yang Anda miliki di Panduan Pengguna IAM.
- Untuk mempelajari cara menyediakan akses ke sumber daya Anda kepada pihak ketiga Akun AWS, lihat Menyediakan akses yang Akun AWS dimiliki oleh pihak ketiga dalam Panduan Pengguna IAM.
- Untuk mempelajari cara memberikan akses melalui federasi identitas, lihat Menyediakan akses ke pengguna terautentikasi eksternal (federasi identitas) dalam Panduan Pengguna IAM.
- Untuk mempelajari perbedaan antara menggunakan peran dan kebijakan berbasis sumber daya untuk akses lintas akun, lihat Akses sumber daya lintas akun di IAM di Panduan Pengguna IAM.

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. Untuk informasi selengkapnya tentang tanggung jawab kepatuhan Anda saat menggunakan Layanan AWS, lihat Dokumentasi AWS Keamanan.

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.

Ketahanan untuk AWS Produk atau Layanan ini

Infrastruktur AWS global dibangun di sekitar Wilayah AWS dan Availability Zones.

Wilayah AWS menyediakan beberapa Availability Zone yang terpisah secara fisik dan terisolasi, yang terhubung dengan latensi rendah, throughput tinggi, dan jaringan yang sangat redundan.

Dengan Zona Ketersediaan, Anda dapat merancang serta mengoperasikan aplikasi dan basis data yang secara otomatis melakukan fail over di antara zona tanpa gangguan. Zona Ketersediaan memiliki ketersediaan dan toleransi kesalahan yang lebih baik, dan dapat diskalakan dibandingkan infrastruktur pusat data tunggal atau multi tradisional.

Untuk informasi selengkapnya tentang AWS Wilayah dan Availability Zone, lihat Infrastruktur AWS Global.

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 panggilan API yang AWS dipublikasikan untuk mengakses AWS Produk atau Layanan ini melalui jaringan. Klien harus mendukung hal-hal berikut:

- Keamanan Lapisan Pengangkutan (TLS). Kami mensyaratkan TLS 1.2 dan menganjurkan TLS 1.3.

- Sandi cocok dengan sistem kerahasiaan maju sempurna (perfect forward secrecy, PFS) seperti DHE (Ephemeral Diffie-Hellman) atau ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Sebagian besar sistem modern seperti Java 7 dan versi lebih baru mendukung mode-mode ini.

Selain itu, permintaan harus ditandatangani menggunakan ID kunci akses dan kunci akses rahasia yang terkait dengan principal IAM. Atau Anda dapat menggunakan AWS Security Token Service (AWS STS) untuk menghasilkan 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.

Menegakkan versi TLS minimum di AWS SDK untuk .NET

Untuk meningkatkan keamanan saat berkomunikasi dengan AWS layanan, Anda harus mengkonfigurasi AWS SDK untuk .NET untuk menggunakan TLS 1.2 atau yang lebih baru.

AWS SDK untuk .NET Menggunakan runtime .NET yang mendasarinya untuk menentukan protokol keamanan mana yang akan digunakan. Secara default, versi.NET saat ini menggunakan protokol terkonfigurasi terbaru yang didukung oleh sistem operasi. Aplikasi Anda dapat mengganti perilaku SDK ini, tetapi tidak disarankan untuk melakukannya.

.NET Core

Secara default, .NET Core menggunakan protokol terkonfigurasi terbaru yang didukung oleh sistem operasi. AWS SDK untuk .NET Tidak menyediakan mekanisme untuk mengesampingkan ini.

Jika Anda menggunakan versi .NET Core lebih awal dari 2.1, kami sangat menyarankan Anda meningkatkan versi.NET Core Anda.

Lihat berikut ini untuk informasi spesifik untuk setiap sistem operasi.

Windows

Distribusi modern Windows memiliki dukungan TLS 1.2 yang diaktifkan secara default. Jika Anda menjalankan Windows 7 SP1 atau Windows Server 2008 R2 SP1, Anda perlu memastikan bahwa dukungan TLS 1.2 diaktifkan di registri, seperti yang dijelaskan di https://learn.microsoft.com/en-us/windows- server/security/tls/tls -registry-settings #tls -12. Jika Anda menjalankan distribusi

sebelumnya, Anda harus meng-upgrade sistem operasi Anda. Untuk informasi tentang dukungan TLS 1.3 di Windows, periksa dokumentasi Microsoft terbaru untuk versi klien atau server minimum yang diperlukan.

macOS

Jika Anda menjalankan .NET Core 2.1 atau yang lebih baru, TLS 1.2 diaktifkan secara default. TLS 1.2 didukung oleh OS X Mavericks v10.9 atau yang lebih baru. .NET Core versi 2.1 dan yang lebih baru memerlukan versi macOS yang lebih baru, seperti yang dijelaskan di? https:// learn.microsoft.com/en-us/ dotnet/core/install/windows tabs=net80&pivots=os-macos.

Jika Anda menggunakan .NET Core 1.0, .NET Core menggunakan OpenSSL di macOS, dependensi yang harus diinstal secara terpisah. OpenSSL menambahkan dukungan untuk TLS 1.2 di versi 1.0.1, dan menambahkan dukungan untuk TLS 1.3 di versi 1.1.1.

Linux

.NET Core di Linux membutuhkan OpenSSL, yang dibundel dengan banyak distribusi Linux. Tapi itu juga bisa dipasang secara terpisah. OpenSSL menambahkan dukungan untuk TLS 1.2 di versi 1.0.1, dan menambahkan dukungan untuk TLS 1.3 di versi 1.1.1. Jika Anda menggunakan versi modern .NET Core (2.1 atau yang lebih baru) dan telah menginstal manajer paket, kemungkinan versi OpenSSL yang lebih modern diinstal untuk Anda.

Yang pasti, Anda dapat menjalankan **`openssl version`**di terminal dan memverifikasi bahwa versinya lebih lambat dari 1.0.1.

.NET Framework

Jika Anda menjalankan versi modern.NET Framework (4.7 atau yang lebih baru) dan versi modern Windows (setidaknya Windows 8 untuk klien, Windows Server 2012 atau yang lebih baru untuk server), TLS 1.2 diaktifkan dan digunakan secara default.

Jika Anda menggunakan runtime .NET Framework yang tidak menggunakan pengaturan sistem operasi (.NET Framework 3.5 hingga 4.5.2), AWS SDK untuk .NET akan mencoba menambahkan dukungan untuk TLS 1.1 dan TLS 1.2 ke protokol yang didukung. Jika Anda menggunakan .NET Framework 3.5, ini akan berhasil hanya jika hot patch yang sesuai diinstal, sebagai berikut:

- Windows 10 versi 1511 dan Windows Server 2016 - KB3156421
- Windows 8.1 dan Windows Server 2012 R2 - KB3154520
- Windows Server 2012 — KB3154519

- Windows 7 SP1 dan Server 2008 R2 SP1 - KB3154518

> **Warning**
>
> Pada 15 Agustus 2024, dukungan AWS SDK untuk .NET berakhir untuk .NET Framework 3.5 dan mengubah versi.NET Framework minimum menjadi 4.7.2. Untuk informasi lebih lanjut, lihat posting blog Perubahan penting yang datang untuk target.NET Framework 3.5 dan 4.5 dari AWS SDK untuk .NET.

Jika aplikasi Anda berjalan pada .NET Framework yang lebih baru pada Windows 7 SP1 atau Windows Server 2008 R2 SP1, Anda perlu memastikan bahwa dukungan TLS 1.2 diaktifkan di registri, seperti yang dijelaskan di https://learn.microsoft.com/en-us/windows- server/security/tls/tls - registry-settings #tls -12. Versi Windows yang lebih baru telah mengaktifkannya secara default.

Untuk praktik terbaik mendetail untuk menggunakan TLS dengan.NET Framework, lihat artikel Microsoft di https://learn.microsoft.com/en-us/dotnet/framework/network-programming/tls.

Alat AWS untuk PowerShell

Alat AWS untuk PowerShellgunakan AWS SDK untuk .NET untuk semua panggilan ke AWS layanan. Perilaku lingkungan Anda tergantung pada versi Windows yang PowerShell Anda jalankan, sebagai berikut.

Windows PowerShell 2.0 hingga 5.x

Windows PowerShell 2.0 hingga 5.x berjalan di .NET Framework. Anda dapat memverifikasi runtime.NET mana (2.0 atau 4.0) yang digunakan PowerShell dengan menggunakan perintah berikut.

```
$PSVersionTable.CLRVersion
```

- Saat menggunakan.NET Runtime 2.0, ikuti petunjuk yang diberikan sebelumnya mengenai AWS SDK untuk .NET dan .NET Framework 3.5.

> **Warning**
>
> Pada 15 Agustus 2024, dukungan AWS SDK untuk .NET berakhir untuk .NET Framework 3.5 dan mengubah versi.NET Framework minimum menjadi 4.7.2. Untuk informasi lebih

lanjut, lihat posting blog Perubahan penting yang datang untuk target.NET Framework 3.5 dan 4.5 dari AWS SDK untuk .NET.

- Saat menggunakan.NET Runtime 4.0, ikuti petunjuk yang diberikan sebelumnya mengenai AWS SDK untuk .NET dan .NET Framework 4+.

Windows PowerShell 6.0

Windows PowerShell 6.0 dan yang lebih baru berjalan di .NET Core. Anda dapat memverifikasi versi.NET Core mana yang digunakan dengan menjalankan perintah berikut.

```
[System.Reflection.Assembly]::GetEntryAssembly().GetCustomAttributes([System.Runtime.Versioning
 $true).FrameworkName
```

Ikuti petunjuk yang diberikan sebelumnya mengenai AWS SDK untuk .NET dan versi yang relevan dari.NET Core.

Xamarin

Untuk Xamarin, lihat petunjuk di https://learn.microsoft.com/en-us/ xamarin/cross-platform/app-fundamentals/transport -layer-security. Ringkasnya:

Untuk Android

- Membutuhkan Android 5.0 atau yang lebih baru.
- Properties Proyek, Opsi Android: HttpClient implementasi harus disetel ke Android dan SSL/TLS implementasinya disetel ke Native TLS 1.2+.

Untuk iOS

- Membutuhkan iOS 7 atau yang lebih baru.
- Properties Project, iOS Build: HttpClient implementasi harus disetel ke NSUrlSession.

Untuk macOS

- Memerlukan macOS 10.9 atau yang lebih baru.
- Project Options, Build, Mac Build: HttpClient implementasi harus diatur ke NSUrlSession.

Unity

Anda harus menggunakan Unity 2018.2 atau yang lebih baru, dan menggunakan runtime scripting .NET 4.x Equivalent. Anda dapat mengatur ini di Pengaturan Proyek, Konfigurasi, Pemain, seperti yang dijelaskan https://docs.unity3d.com/2019.1/Documentation/Manual/ScriptingRuntimeUpgradedi.html. Runtime scripting Setara .NET 4.x memungkinkan dukungan TLS 1.2 untuk semua platform Unity yang menjalankan Mono atau CPP. IL2

Browser (untuk Blazor WebAssembly)

WebAssembly berjalan di browser bukan di server, dan menggunakan browser untuk menangani lalu lintas HTTP. Oleh karena itu, dukungan TLS ditentukan oleh dukungan browser.

Blazor WebAssembly, dalam pratinjau untuk ASP.NET Core 3.1, hanya didukung di browser yang mendukung WebAssembly, seperti yang dijelaskan di -platform. https://learn.microsoft.com/en-us/aspnet/core/blazor/supported Semua browser mainstream mendukung TLS 1.2 sebelum mendukung. WebAssembly Jika ini adalah kasus untuk browser Anda, maka jika aplikasi Anda berjalan, itu dapat berkomunikasi melalui TLS 1.2.

Lihat dokumentasi browser Anda untuk informasi dan verifikasi selengkapnya.

Migrasi Klien Enkripsi S3 (V1 ke V2)

> **Note**
>
> Jika Anda menggunakan V2 dan ingin bermigrasi ke V4, lihat. Migrasi Klien Enkripsi S3 (V2 ke V4)

Topik ini menunjukkan cara memigrasikan aplikasi dari klien enkripsi Amazon Simple Storage Service (S3) Versi 1 (V1) ke Versi 2 (V2), dan memastikan ketersediaan aplikasi selama proses migrasi.

Objek yang dienkripsi dengan klien V2 tidak dapat didekripsi dengan klien V1. Untuk memudahkan migrasi ke klien baru tanpa harus mengenkripsi ulang semua objek sekaligus, klien “V1-transisi” telah disediakan. Klien ini dapat mendekripsi objek terenkripsi V1- dan V2, tetapi mengenkripsi objek hanya dalam format yang kompatibel dengan V1. Klien V2 dapat mendekripsi objek terenkripsi V1- dan V2 (bila diaktifkan untuk objek V1), tetapi mengenkripsi objek hanya dalam format yang kompatibel dengan V2.

Ikhtisar Migrasi

Migrasi ini terjadi dalam tiga fase. Fase-fase ini diperkenalkan di sini dan dijelaskan secara rinci nanti. Setiap fase harus diselesaikan untuk semua klien yang menggunakan objek bersama sebelum fase berikutnya dimulai.

1. Perbarui klien yang ada ke klien transisi V1 untuk membaca format baru. Pertama, perbarui aplikasi Anda untuk mengambil ketergantungan pada klien transisi V1 alih-alih klien V1. Klien transisi V1 memungkinkan kode Anda yang ada untuk mendekripsi objek yang ditulis oleh klien V2 baru dan objek yang ditulis dalam format yang kompatibel dengan V1.

   > **Note**
   >
   > Klien transisi V1 disediakan hanya untuk tujuan migrasi. Lanjutkan untuk memutakhirkan ke klien V2 setelah pindah ke klien transisi V1.

2. Migrasikan klien transisi V1 ke klien V2 untuk menulis format baru. Selanjutnya, ganti semua klien transisi V1 di aplikasi Anda dengan klien V2, dan atur profil keamanannya. `V2AndLegacy` Menyetel profil keamanan ini pada klien V2 memungkinkan klien tersebut untuk mendekripsi objek yang dienkripsi dalam format yang kompatibel dengan V1.
3. Perbarui klien V2 agar tidak lagi membaca format V1. Akhirnya, setelah semua klien dimigrasikan ke V2 dan semua objek telah dienkripsi atau dienkripsi ulang dalam format yang kompatibel dengan V2, atur profil keamanan V2 menjadi alih-alih. `V2` `V2AndLegacy` Ini mencegah dekripsi objek yang dalam format yang kompatibel dengan V1.

Perbarui Klien yang Ada ke Klien Transisi V1 untuk Membaca Format Baru

Klien enkripsi V2 menggunakan algoritma enkripsi yang tidak didukung oleh versi klien yang lebih lama. Langkah pertama dalam migrasi adalah memperbarui klien dekripsi V1 Anda sehingga mereka dapat membaca format baru.

Klien V1-transisi memungkinkan aplikasi Anda untuk mendekripsi objek terenkripsi V1 dan V2. Klien ini adalah bagian dari paket NuGet Amazon.Extensions.S3.Encryption. Lakukan langkah-langkah berikut pada setiap aplikasi Anda untuk menggunakan klien transisi V1.

1. Ambil ketergantungan baru pada paket Amazon.Extensions.S3.Encryption. Jika proyek Anda bergantung langsung pada AWSSDK.S3 atau. AWSSDK KeyManagementServicepaket, Anda

harus memperbarui dependensi tersebut atau menghapusnya sehingga versi yang diperbarui akan ditarik dengan paket baru ini.

2. Ubah `using` pernyataan yang sesuai dari `Amazon.S3.Encryption` menjadi`Amazon.Extensions.S3.Encryption`, sebagai berikut:

```
// using Amazon.S3.Encryption;
  using Amazon.Extensions.S3.Encryption;
```

3. Membangun kembali dan menerapkan kembali aplikasi Anda.

Klien transisi V1 sepenuhnya kompatibel dengan API dengan klien V1, jadi tidak ada perubahan kode lain yang diperlukan.

Migrasikan Klien Transisi V1 ke Klien V2 untuk Menulis Format Baru

Klien V2 adalah bagian dari paket NuGet Amazon.Extensions.S3.Encryption. Ini memungkinkan aplikasi Anda untuk mendekripsi objek terenkripsi V1- dan V2 (jika dikonfigurasi untuk melakukannya), tetapi mengenkripsi objek hanya dalam format yang kompatibel dengan V2.

Setelah memperbarui klien Anda yang ada untuk membaca format enkripsi baru, Anda dapat melanjutkan untuk memperbarui aplikasi Anda dengan aman ke klien enkripsi dan dekripsi V2. Lakukan langkah-langkah berikut pada setiap aplikasi Anda untuk menggunakan klien V2:

1. Ubah `EncryptionMaterials` ke `EncryptionMaterialsV2`.

   a. Saat menggunakan KMS:

      i. Berikan ID kunci KMS.

      ii. Deklarasikan metode enkripsi yang Anda gunakan; yaitu,. `KmsType.KmsContext`

      iii. Berikan konteks enkripsi ke KMS untuk dikaitkan dengan kunci data ini. Anda dapat mengirim kamus kosong (konteks enkripsi Amazon masih akan digabungkan), tetapi memberikan konteks tambahan dianjurkan.

   b. Saat menggunakan metode pembungkus kunci yang disediakan pengguna (enkripsi simetris atau asimetris):

      i. Menyediakan `AES` atau `RSA` contoh yang berisi materi enkripsi.

ii. Deklarasikan algoritma enkripsi mana yang akan digunakan; yaitu, `SymmetricAlgorithmType.AesGcm` atau. `AsymmetricAlgorithmType.RsaOaepSha1`

2. Ubah `AmazonS3CryptoConfiguration` ke `AmazonS3CryptoConfigurationV2` dengan `SecurityProfile` properti disetel ke`SecurityProfile.V2AndLegacy`.
3. Ubah `AmazonS3EncryptionClient` ke `AmazonS3EncryptionClientV2`. Klien ini mengambil yang baru dikonversi `AmazonS3CryptoConfigurationV2` dan `EncryptionMaterialsV2` objek dari langkah sebelumnya.

Contoh: Konteks KMS ke KMS +

Pra-migrasi

```
using System.Security.Cryptography;
using Amazon.S3.Encryption;

var encryptionMaterial = new
 EncryptionMaterials("1234abcd-12ab-34cd-56ef-1234567890ab");
var configuration = new AmazonS3CryptoConfiguration()
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
var encryptionClient = new AmazonS3EncryptionClient(configuration, encryptionMaterial);
```

Pasca-migrasi

```
using System.Security.Cryptography;
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;

var encryptionContext = new Dictionary<string, string>();
var encryptionMaterial = new
 EncryptionMaterialsV2("1234abcd-12ab-34cd-56ef-1234567890ab", KmsType.KmsContext,
 encryptionContext);
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy)
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
var encryptionClient = new AmazonS3EncryptionClientV2(configuration,
 encryptionMaterial);
```

Contoh: Algoritma Simetris (AES-CBC ke AES-GCM Key Wrap)

`StorageMode`bisa salah satu `ObjectMetadata` atau`InstructionFile`.

Pra-migrasi

```
using System.Security.Cryptography;
using Amazon.S3.Encryption;

var symmetricAlgorithm = Aes.Create();
var encryptionMaterial = new EncryptionMaterials(symmetricAlgorithm);
var configuration = new AmazonS3CryptoConfiguration()
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
var encryptionClient = new AmazonS3EncryptionClient(configuration, encryptionMaterial);
```

Pasca-migrasi

```
using System.Security.Cryptography;
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;

var symmetricAlgorithm = Aes.Create();
var encryptionMaterial = new EncryptionMaterialsV2(symmetricAlgorithm,
 SymmetricAlgorithmType.AesGcm);
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy)
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
var encryptionClient = new AmazonS3EncryptionClientV2(configuration,
 encryptionMaterial);
```

> **Note**
>
> Saat mendekripsi dengan AES-GCM, baca seluruh objek sampai akhir sebelum Anda mulai menggunakan data yang didekripsi. Ini untuk memverifikasi bahwa objek belum dimodifikasi sejak dienkripsi.

Contoh: Algoritma Asimetris (RSA ke RSA-OAEP-SHA 1 Key Wrap)

`StorageMode`bisa salah satu `ObjectMetadata` atau`InstructionFile`.

Pra-migrasi

```
using System.Security.Cryptography;
using Amazon.S3.Encryption;

var asymmetricAlgorithm = RSA.Create();
var encryptionMaterial = new EncryptionMaterials(asymmetricAlgorithm);
var configuration = new AmazonS3CryptoConfiguration()
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
var encryptionClient = new AmazonS3EncryptionClient(configuration, encryptionMaterial);
```

Pasca-migrasi

```
using System.Security.Cryptography;
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;

var asymmetricAlgorithm = RSA.Create();
var encryptionMaterial = new EncryptionMaterialsV2(asymmetricAlgorithm,
 AsymmetricAlgorithmType.RsaOaepSha1);
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy)
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
var encryptionClient = new AmazonS3EncryptionClientV2(configuration,
 encryptionMaterial);
```

Perbarui Klien V2 agar Tidak Lagi Membaca Format V1

Akhirnya, semua objek akan dienkripsi atau dienkripsi ulang menggunakan klien V2. Setelah konversi ini selesai, Anda dapat menonaktifkan kompatibilitas V1 di klien V2 dengan menyetel `SecurityProfile` properti ke`SecurityProfile.V2`, seperti yang ditunjukkan pada cuplikan berikut.

```
//var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy);
```

```
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2);
```

Migrasi Klien Enkripsi S3 (V2 ke V4)

> **Note**
>
> Jika Anda menggunakan V1 dan ingin bermigrasi ke V4, Anda harus terlebih dahulu bermigrasi ke V2. Lihat Migrasi Klien Enkripsi S3 (V1 ke V2).

Topik ini menunjukkan cara memigrasikan aplikasi Anda dari klien enkripsi Amazon Simple Storage Service (S3) Versi 2 (V4) ke Versi 2 (V4), dan memastikan ketersediaan aplikasi selama proses migrasi. V4 menggunakan AES-GCM dengan komitmen utama untuk enkripsi konten dan memperkenalkan Kebijakan Komitmen untuk meningkatkan keamanan terhadap serangan substitusi kunci.

Klien V4 tersedia dalam paket NuGet Amazon.Extensions.S3.Encryption.

> **Important**
>
> Melanggar Perubahan: Klien V4 yang dikonfigurasi dengan `RequireEncryptRequireDecrypt` kebijakan tidak dapat mendekripsi objek yang dienkripsi dengan klien V1 atau V2. Hanya klien V2 terbaru yang dapat mendekripsi objek terenkripsi V4 dengan komitmen utama. Sebelum beralih ke `RequireEncryptRequireDecrypt` kebijakan, Anda harus mengenkripsi ulang semua data yang ada menggunakan klien V4 dengan komitmen utama diaktifkan.

Memahami Konsep V4

V4 menggunakan AES-GCM dengan komitmen utama untuk enkripsi konten dan memperkenalkan konsep keamanan utama yang meningkatkan perlindungan data terenkripsi Anda:

Kebijakan Komitmen

Kebijakan Komitmen mengontrol cara klien enkripsi menangani komitmen utama selama operasi enkripsi dan dekripsi. V4 mendukung tiga kebijakan komitmen:

`ForbidEncryptAllowDecrypt`

Enkripsi: Tanpa komitmen

Dekripsi: Memungkinkan objek yang tidak berkomitmen

Keamanan: Tidak menegakkan komitmen dan memungkinkan gangguan

Kompatibilitas: Semua implementasi V2 dan V4 dapat membaca objek yang dienkripsi dengan kebijakan ini

`RequireEncryptAllowDecrypt`

Enkripsi: Dengan komitmen utama

Dekripsi: Memungkinkan objek yang melakukan dan tidak berkomitmen

Keamanan: Objek baru dilindungi dari serangan substitusi kunci, objek lama masih dapat dibaca

Kompatibilitas: Hanya V4 yang mendukung kebijakan ini

`RequireEncryptRequireDecrypt`(Default untuk V4)

Enkripsi: Dengan komitmen utama

Dekripsi: Hanya melakukan objek

Keamanan: Penegakan komitmen penuh untuk keamanan maksimum

Kompatibilitas: Hanya V4 yang mendukung kebijakan ini

AES GCM dengan Komitmen Utama

V4 menggunakan AES-GCM dengan komitmen utama untuk enkripsi konten, yang memberikan keamanan yang ditingkatkan:

- Perlindungan Perusakan: Ini melindungi terhadap serangan substitusi kunci dengan mengikat kunci ke data terenkripsi secara kriptografis.
- Kompatibilitas Versi: Objek yang dienkripsi dengan komitmen utama hanya dapat didekripsi oleh klien V4 dan versi yang lebih baru.

> **Warning**
>
> Sebelum mengaktifkan enkripsi komitmen utama dalam produksi, pastikan bahwa semua aplikasi yang perlu mendekripsi objek Anda telah ditingkatkan ke V4 atau yang lebih baru karena klien V2 tidak digunakan lagi.

Perbarui Klien yang Ada untuk Membaca Format V4

Klien enkripsi V4 menggunakan algoritma enkripsi yang tidak didukung oleh versi klien yang lebih lama. Langkah pertama dalam migrasi adalah memperbarui klien V2 Anda sehingga mereka dapat membaca format V4 baru.

Perbarui Dependensi NuGet Package

Perbarui aplikasi Anda untuk menggunakan versi terbaru dari paket NuGet Amazon.Extensions.S3.Encryption yang mencakup dukungan V4. Lakukan langkah-langkah berikut pada setiap aplikasi Anda:

1. Perbarui ke paket Amazon.Extensions.S3.Encryption terbaru. Jika proyek Anda bergantung langsung pada AWSSDK.S3 atau. AWSSDK KeyManagementServicepaket, Anda harus memperbarui dependensi tersebut atau menghapusnya sehingga versi yang diperbarui akan ditarik dengan paket baru ini.
2. Pastikan `using` pernyataan Anda mereferensikan namespace yang benar:

   ```
   using Amazon.Extensions.S3.Encryption;
   using Amazon.Extensions.S3.Encryption.Primitives;
   ```

3. Membangun kembali dan menerapkan kembali aplikasi Anda.

Klien V2 Anda yang ada akan terus bekerja dengan paket yang diperbarui dan akan dapat mendekripsi objek yang dienkripsi oleh klien V4 (tergantung pada kebijakan komitmen yang digunakan).

Membangun dan Menyebarkan Aplikasi

Setelah memperbarui dependensi NuGet paket Anda:

1. Bangun aplikasi Anda untuk memastikan semua dependensi diselesaikan dengan benar.

2. Uji aplikasi Anda di lingkungan pengembangan untuk memverifikasi bahwa fungsionalitas yang ada terus berfungsi.
3. Terapkan aplikasi yang diperbarui ke lingkungan produksi Anda.

Pembaruan ini memungkinkan klien V2 Anda yang ada untuk mendekripsi objek yang akan dienkripsi oleh klien V4, memastikan kompatibilitas selama proses migrasi.

Migrasi ke Klien V4

Setelah memperbarui klien Anda yang ada untuk membaca format enkripsi baru, Anda dapat melanjutkan untuk memperbarui aplikasi Anda dengan aman untuk menggunakan enkripsi V4 dan klien dekripsi. Klien V4 memberikan keamanan yang ditingkatkan melalui komitmen utama sambil mempertahankan kompatibilitas dengan objek terenkripsi yang ada.

Proses Migrasi 4 Langkah

Migrasi dari V2 ke V4 mengikuti proses 4 langkah terstruktur untuk memastikan kompatibilitas dan keamanan. Setiap langkah mewakili konfigurasi spesifik yang harus diterapkan di semua aplikasi Anda sebelum melanjutkan ke langkah berikutnya.

1. Langkah 0: Klien V2 (Titik Awal) - Implementasi V2 Anda yang ada
2. Langkah 1: V4 dengan Kompatibilitas V2 - Migrasi ke klien V4 sambil mempertahankan perilaku enkripsi yang kompatibel dengan V2
3. Langkah 2: V4 dengan Key Commitment Writes - Mulai mengenkripsi dengan komitmen utama sambil memungkinkan dekripsi objek warisan
4. Langkah 3: V4 dengan Penegakan Penuh - Memerlukan komitmen utama untuk enkripsi dan dekripsi

Langkah 0: Klien V2 (Titik Awal)

Ini mewakili konfigurasi klien V2 Anda yang ada. Langkah ini menunjukkan status awal sebelum migrasi.

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;
using Amazon.S3.Model;

// Step 0: V2 Client - Starting configuration
```

```
var encryptionContext = new Dictionary<string, string>();
var encryptionMaterial = new EncryptionMaterialsV2(kmsKeyId, KmsType.KmsContext,
 encryptionContext);

#pragma warning disable 0618
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2);
#pragma warning enable 0618

var encryptionClient = new AmazonS3EncryptionClientV2(configuration,
 encryptionMaterial);

// Use the client for PutObject and GetObject operations
await encryptionClient.PutObjectAsync(new PutObjectRequest
{
    BucketName = bucket,
    Key = objectKey,
    ContentBody = content
});
```

Langkah 1: V4 dengan Kompatibilitas V2

Migrasi ke klien V4 sambil mempertahankan perilaku identik ke V2. Langkah ini menggunakan `ForbidEncryptAllowDecrypt` kebijakan untuk mengenkripsi tanpa komitmen dan memungkinkan dekripsi semua objek.

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;
using Amazon.S3.Model;

// Step 1: V4 Client with V2 compatibility
var encryptionContext = new Dictionary<string, string>();
var encryptionMaterial = new EncryptionMaterialsV4(kmsKeyId, KmsType.KmsContext,
 encryptionContext);

var configuration = new AmazonS3CryptoConfigurationV4(
    SecurityProfile.V4,
    CommitmentPolicy.ForbidEncryptAllowDecrypt,
    ContentEncryptionAlgorithm.AesGcm);

var encryptionClient = new AmazonS3EncryptionClientV4(configuration,
 encryptionMaterial);

// Use the client for PutObject and GetObject operations
```

```
await encryptionClient.PutObjectAsync(new PutObjectRequest
{
    BucketName = bucket,
    Key = objectKey,
    ContentBody = content
});
```

Perilaku: Mengenkripsi tanpa komitmen, dapat mendekripsi objek yang berkomitmen dan tidak berkomitmen. Identik dengan perilaku V2.

Langkah 2: V4 dengan Key Commitment Writes

Mulai enkripsi dengan komitmen utama sambil mempertahankan kompatibilitas mundur untuk dekripsi. Langkah ini menggunakan `RequireEncryptAllowDecrypt` kebijakan.

> **Warning**
>
> Sebelum menerapkan Langkah 2, pastikan semua pembaca telah diperbarui ke Langkah 1 atau yang lebih baru untuk menangani enkripsi komitmen utama.

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;
using Amazon.S3.Model;

// Step 2: V4 Client with key commitment writes
var encryptionContext = new Dictionary<string, string>();
var encryptionMaterial = new EncryptionMaterialsV4(kmsKeyId, KmsType.KmsContext,
 encryptionContext);

var configuration = new AmazonS3CryptoConfigurationV4(
    SecurityProfile.V4,
    CommitmentPolicy.RequireEncryptAllowDecrypt,
    ContentEncryptionAlgorithm.AesGcmWithCommitment);

var encryptionClient = new AmazonS3EncryptionClientV4(configuration,
 encryptionMaterial);

// Use the client for PutObject and GetObject operations
await encryptionClient.PutObjectAsync(new PutObjectRequest
{
    BucketName = bucket,
```

```
    Key = objectKey,
    ContentBody = content
});
```

Perilaku: Mengenkripsi dengan komitmen, dapat mendekripsi objek yang berkomitmen dan tidak berkomitmen. Objek baru dilindungi dari serangan substitusi kunci.

Langkah 3: V4 dengan Penegakan Penuh

Memerlukan komitmen utama untuk enkripsi dan dekripsi. Langkah ini menggunakan `RequireEncryptRequireDecrypt` kebijakan untuk keamanan maksimum.

> **Warning**
>
> Sebelum menerapkan Langkah 3, pastikan semua objek di sistem Anda telah dienkripsi ulang dengan komitmen utama (Langkah 2). Langkah ini akan gagal mendekripsi objek yang dienkripsi tanpa komitmen.

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;
using Amazon.S3.Model;

// Step 3: V4 Client with full key commitment enforcement
var encryptionContext = new Dictionary<string, string>();
var encryptionMaterial = new EncryptionMaterialsV4(kmsKeyId, KmsType.KmsContext,
 encryptionContext);

var configuration = new AmazonS3CryptoConfigurationV4(
    SecurityProfile.V4,
    CommitmentPolicy.RequireEncryptRequireDecrypt,
    ContentEncryptionAlgorithm.AesGcmWithCommitment);

var encryptionClient = new AmazonS3EncryptionClientV4(configuration,
 encryptionMaterial);

// Use the client for PutObject and GetObject operations
await encryptionClient.PutObjectAsync(new PutObjectRequest
{
    BucketName = bucket,
    Key = objectKey,
```

```
    ContentBody = content
});
```

Perilaku: Mengenkripsi dengan komitmen, hanya mendekripsi objek yang dienkripsi dengan komitmen. Keamanan maksimum terhadap serangan substitusi kunci.

Contoh Konfigurasi Tambahan

Bagian ini memberikan contoh tambahan untuk mengonfigurasi klien V4 dengan opsi berbeda selama migrasi.

Mengaktifkan Legacy Support

Untuk mengaktifkan klien V4 membaca objek yang dienkripsi oleh klien V1 dan V2, konfigurasikan klien dengan kebijakan komitmen yang memungkinkan dekripsi lama:

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;

// Configure V4 client to read V1/V2 objects
var configuration = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4AndLegacy,
 CommitmentPolicy.RequireEncryptAllowDecrypt)
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};

// This configuration allows:
// - Encryption: With commitment (secure)
// - Decryption: Both V2 (non-committing) and V4 (committing) objects
```

Gunakan konfigurasi ini selama migrasi saat Anda perlu mendekripsi objek yang dienkripsi oleh klien lama sambil memastikan objek baru dienkripsi dengan keamanan yang ditingkatkan.

Mengkonfigurasi Metode Penyimpanan

V4 mendukung dua metode penyimpanan untuk metadata enkripsi. Pilih metode yang paling sesuai dengan kasus penggunaan Anda:

Metadata Objek (Default)

```
var configuration = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.RequireEncryptAllowDecrypt)
```

```
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
// Encryption metadata is stored in S3 object metadata
```

File Instruksi

```
var configuration = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.RequireEncryptAllowDecrypt)
{
    StorageMode = CryptoStorageMode.InstructionFile
};
// Encryption metadata is stored in a separate S3 object (instruction file)
```

Gunakan `InstructionFile` saat Anda perlu melestarikan metadata objek untuk tujuan lain atau saat bekerja dengan objek yang memiliki batasan ukuran metadata.

Mengkonfigurasi Kebijakan Komitmen

Pilih kebijakan komitmen yang sesuai berdasarkan persyaratan keamanan dan fase migrasi Anda:

Fase Migrasi (Kompatibilitas V2)

```
// For migration: encrypt without commitment, allow all decryption
var migrationConfig = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.ForbidEncryptAllowDecrypt);
```

Fase Transisi (Direkomendasikan)

```
// For transition: encrypt with commitment, allow legacy decryption
var transitionConfig = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.RequireEncryptAllowDecrypt);
```

Fase Keamanan Penuh

```
// For maximum security: require commitment for both encryption and decryption
var secureConfig = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.RequireEncryptRequireDecrypt);
```

Mulailah dengan `ForbidEncryptAllowDecrypt` selama migrasi awal, pindah ke `RequireEncryptAllowDecrypt` untuk fase transisi, dan akhirnya gunakan

`RequireEncryptRequireDecrypt` ketika semua klien telah ditingkatkan dan semua objek telah dienkripsi ulang dengan komitmen.

Pertimbangan khusus untuk AWS SDK untuk .NET

Bagian ini berisi pertimbangan untuk kasus khusus di mana konfigurasi atau prosedur normal tidak sesuai atau memadai.

Topik

- Memperoleh majelis untuk AWS SDK untuk .NET
- Mengakses kredensi dan profil dalam aplikasi
- Pertimbangan khusus untuk dukungan Unity
- Pertimbangan khusus untuk dukungan Xamarin

Memperoleh majelis untuk AWS SDK untuk .NET

Topik ini menjelaskan bagaimana Anda dapat memperoleh AWSSDK rakitan dan menyimpannya secara lokal (atau di tempat) untuk digunakan dalam proyek Anda. Ini bukan metode yang direkomendasikan untuk menangani referensi SDK, tetapi diperlukan di beberapa lingkungan.

> **Note**
>
> Metode yang disarankan untuk menangani referensi SDK adalah mengunduh dan menginstal hanya NuGet paket yang dibutuhkan setiap proyek. Metode itu dijelaskan dalamInstal AWSSDK paket dengan NuGet.

Jika Anda tidak dapat atau tidak diizinkan untuk mengunduh dan menginstal NuGet paket per proyek, opsi berikut tersedia untuk Anda.

Unduh dan ekstrak file ZIP

(Ingat bahwa ini bukan metode yang direkomendasikan untuk menangani referensi ke AWS SDK untuk .NET.)

1. Unduh salah satu file ZIP berikut:

   - aws-sdk-net8.0.zip - Majelis yang mendukung .NET 8 dan yang lebih baru.
   - aws-sdk-netcoreapp3.1.zip - Majelis yang mendukung .NET Core 3.1 dan yang lebih baru.

- aws-sdk-netstandard2.0.zip - Majelis yang mendukung .NET Standard 2.0 dan 2.1.
- aws-sdk-net472.zip - Majelis yang mendukung .NET Framework 4.5 dan yang lebih baru.

2. Ekstrak rakitan ke beberapa folder “unduh” di sistem file Anda; tidak masalah di mana. Catat folder ini.
3. Ketika Anda mengatur proyek Anda, Anda mendapatkan rakitan yang diperlukan dari folder ini, seperti yang dijelaskan dalam. Instal AWSSDK rakitan tanpa NuGet

Mengakses kredensi dan profil dalam aplikasi

Metode yang lebih disukai untuk menggunakan kredensional adalah memungkinkan AWS SDK untuk .NET untuk menemukan dan mengambilnya untuk Anda, seperti yang dijelaskan dalam. Resolusi kredensi dan profil

Namun, Anda juga dapat mengonfigurasi aplikasi Anda untuk secara aktif mengambil profil dan kredensional, dan kemudian secara eksplisit menggunakan kredensional tersebut saat membuat klien layanan. AWS

Untuk secara aktif mengambil profil dan kredensional, gunakan kelas dari Amazon.Runtime.CredentialManagementnamespace.

- Untuk menemukan profil dalam file yang menggunakan format file AWS kredensial (baik file AWS kredensial bersama di lokasi default atau file kredensial kustom), gunakan kelas. SharedCredentialsFile File dalam format ini kadang-kadang hanya disebut file kredensional dalam teks ini untuk singkatnya.
- Untuk menemukan profil di SDK Store, gunakan kelas Net SDKCredentials File.
- Untuk mencari di kedua file kredensial dan SDK Store, tergantung pada konfigurasi properti kelas, gunakan kelas. CredentialProfileStoreChain

  Anda dapat menggunakan kelas ini untuk menemukan profil. Anda juga dapat menggunakan kelas ini untuk meminta AWS kredensional secara langsung alih-alih menggunakan `AWSCredentialsFactory` kelas (dijelaskan selanjutnya).
- Untuk mengambil atau membuat berbagai jenis kredensional dari profil, gunakan kelas Factory. AWSCredentials

Bagian berikut memberikan contoh untuk kelas-kelas ini.

Contoh untuk kelas CredentialProfileStoreChain

Anda bisa mendapatkan kredensi atau profil dari CredentialProfileStoreChainkelas dengan menggunakan metode TryGetAWSCredentialsor TryGetProfile. `ProfilesLocation`Properti kelas menentukan perilaku metode, sebagai berikut:

- Jika `ProfilesLocation` nol atau kosong, cari SDK Store jika platform mendukungnya, lalu cari file AWS kredensial bersama di lokasi default.
- Jika `ProfilesLocation` properti berisi nilai, cari file kredensial yang ditentukan dalam properti.

Mendapatkan kredensional dari SDK Store atau file kredensial bersama AWS

Contoh ini menunjukkan kepada Anda cara mendapatkan kredensi dengan menggunakan `CredentialProfileStoreChain` kelas dan kemudian menggunakan kredensialnya untuk membuat objek Amazons3Client. Kredensional dapat berasal dari SDK Store atau dari file AWS kredensional bersama di lokasi default.

Contoh ini juga menggunakan Amazon.Runtime. AWSCredentialskelas.

```
var chain = new CredentialProfileStoreChain();
AWSCredentials awsCredentials;
if (chain.TryGetAWSCredentials("some_profile", out awsCredentials))
{
    // Use awsCredentials to create an Amazon S3 service client
    using (var client = new AmazonS3Client(awsCredentials))
    {
        var response = await client.ListBucketsAsync();
        Console.WriteLine($"Number of buckets: {response.Buckets.Count}");
    }
}
```

Mendapatkan profil dari SDK Store atau file AWS kredensial bersama

Contoh ini menunjukkan kepada Anda cara mendapatkan profil dengan menggunakan CredentialProfileStoreChain kelas. Kredensional dapat berasal dari SDK Store atau dari file AWS kredensional bersama di lokasi default.

Contoh ini juga menggunakan CredentialProfilekelas.

```
var chain = new CredentialProfileStoreChain();
```

```
CredentialProfile basicProfile;
if (chain.TryGetProfile("basic_profile", out basicProfile))
{
    // Use basicProfile
}
```

Dapatkan kredensional dari file kredensial kustom

Contoh ini menunjukkan kepada Anda cara mendapatkan kredensi dengan menggunakan kelas. CredentialProfileStoreChain Kredensialnya berasal dari file yang menggunakan format file AWS kredensial tetapi berada di lokasi alternatif.

Contoh ini juga menggunakan Amazon.Runtime. AWSCredentialskelas.

```
var chain = new
    CredentialProfileStoreChain("c:\\Users\\sdkuser\\customCredentialsFile.ini");
AWSCredentials awsCredentials;
if (chain.TryGetAWSCredentials("basic_profile", out awsCredentials))
{
    // Use awsCredentials to create an AWS service client
}
```

Contoh untuk kelas SharedCredentialsFile dan AWSCredentials Pabrik

Buat Amazons3Client dengan menggunakan kelas SharedCredentialsFile

Contoh ini menunjukkan kepada Anda cara menemukan profil di file AWS kredensial bersama, membuat kredensi dari profil, dan kemudian menggunakan AWS kredensialnya untuk membuat objek Amazons3Client. Contoh menggunakan SharedCredentialsFilekelas.

Contoh ini juga menggunakan CredentialProfilekelas dan Amazon.Runtime. AWSCredentialskelas.

```
CredentialProfile basicProfile;
AWSCredentials awsCredentials;
var sharedFile = new SharedCredentialsFile();
if (sharedFile.TryGetProfile("basic_profile", out basicProfile) &&
    AWSCredentialsFactory.TryGetAWSCredentials(basicProfile, sharedFile, out
 awsCredentials))
{
    // use awsCredentials to create an Amazon S3 service client
    using (var client = new AmazonS3Client(awsCredentials, basicProfile.Region))
```

```
    {
        var response = await client.ListBucketsAsync();
        Console.WriteLine($"Number of buckets: {response.Buckets.Count}");
    }
}
```

> **Note**
>
> Kelas Net SDKCredentials File dapat digunakan dengan cara yang persis sama, kecuali Anda akan membuat instance objek Net SDKCredentials File baru alih-alih objek. SharedCredentialsFile

Pertimbangan khusus untuk dukungan Unity

Saat menggunakan AWS SDK untuk .NET dan .NET Standard 2.0 untuk aplikasi Unity Anda, aplikasi Anda harus mereferensikan AWS SDK untuk .NET rakitan (file DLL) secara langsung daripada menggunakan. NuGet Mengingat persyaratan ini, berikut ini adalah tindakan penting yang perlu Anda lakukan.

- Anda perlu mendapatkan AWS SDK untuk .NET majelis dan menerapkannya pada proyek Anda. Untuk informasi tentang cara melakukan ini, lihat Unduh dan ekstrak file ZIP di topikMemperoleh AWSSDK majelis.

- Anda perlu menyertakan yang berikut ini DLLs dalam proyek Unity Anda bersama DLLs for AWSSDK.Core dan AWS layanan lain yang Anda gunakan. Dimulai dengan versi 3.5.109 dari AWS SDK untuk .NET, file ZIP Standar .NET berisi tambahan ini. DLLs
  - Microsoft.Bcl. AsyncInterfaces.dll
  - System.Runtime. CompilerServices.unsafe.dll
  - System.threading.tasks.extensions.dll

- Jika Anda menggunakan IL2CPP untuk membangun proyek Unity Anda, Anda harus menambahkan `link.xml` file ke folder Asset Anda untuk mencegah pengupasan kode. `link.xml`File harus mencantumkan semua AWSSDK rakitan yang Anda gunakan, dan masing-masing harus menyertakan atribut. `preserve="all"` Cuplikan berikut menunjukkan contoh file ini.

```
<linker>
    <assembly fullname="AWSSDK.Core" preserve="all"/>
    <assembly fullname="AWSSDK.DynamoDBv2" preserve="all"/>
    <assembly fullname="AWSSDK.Lambda" preserve="all"/>
</linker>
```

> **Note**
>
> Untuk membaca informasi latar belakang yang menarik terkait dengan persyaratan ini, lihat artikel di https://aws.amazon.com/blogs/developer/ referencing-the-aws-sdk - for-net-standard -2-0- -uwp/. from-unity-xamarin-or

Pertimbangan khusus untuk dukungan Xamarin

Proyek Xamarin (baru dan yang sudah ada) harus menargetkan .NET Standard 2.0. Lihat Dukungan .NET Standard 2.0 di Xamarin.Forms dan dukungan implementasi.NET.

Lihat juga informasi tentang Portable Class Library dan Xamarin.

Referensi API untuk AWS SDK untuk .NET

AWS SDK untuk .NET Ini menyediakan API yang dapat Anda gunakan untuk mengakses AWS layanan. Untuk melihat kelas dan metode apa yang tersedia di API, lihat Referensi AWS SDK untuk .NET API.

Selain referensi umum yang diberikan di atas, masing-masing contoh di bawah Contoh kode terpandu bagian berisi referensi ke metode dan kelas tertentu yang digunakan dalam contoh itu.

Tentang versi referensi API

Referensi API yang dijelaskan sebelumnya adalah untuk versi 3.0 dan yang lebih baru AWS SDK untuk .NET.

Untuk informasi tentang migrasi dari versi SDK yang lebih lama, lihat Migrasi proyek Anda

Untuk menemukan konten usang untuk referensi SDK API versi sebelumnya, lihat item berikut:

- AWS SDK untuk .NET Referensi API V1 (usang)
- AWS SDK untuk .NET Referensi API V2 (usang)

Untuk melihat referensi AWS SDK untuk .NET API yang tidak digunakan lagi, Anda harus mengekstraknya dan mengonfigurasi browser web. Instruksi yang ditunjukkan selanjutnya adalah contoh bagaimana melakukan ini. Mereka didasarkan pada penggunaan browser web Google Chrome pada sistem Windows. Sesuaikan dengan browser web dan sistem operasi spesifik Anda.

Lihat Referensi API V1 yang tidak digunakan lagi

1. Unduh file ZIP untuk Referensi AWS SDK untuk .NET API V1 yang tidak digunakan lagi. Secara default, nama file ZIP adalah`sdkfornet-api-ref_v1_deprecated.zip`. Nama itu akan digunakan di seluruh instruksi ini.
2. Tempatkan file ZIP di folder pilihan Anda. Untuk instruksi ini, nama folder diasumsikan`C:\work\temp\api-refs\V1`.
3. Klik kanan pada file ZIP dan pilih Extract All. Terima lokasi default, yang `C:\work\temp\api-refs\V1\sdkfornet-api-ref_v1` untuk instruksi ini.

4. Buat pintasan untuk Google Chrome di `C:\work\temp\api-refs\V1` folder. Berhati-hatilah untuk tidak memindahkan aplikasi asli.
5. Di properti pintasan baru, atur bidang berikut:

   - Target: "*<path to the Google Chrome application>*" `--disable-web-security --user-data-dir=C:\work\temp\api-refs\V1\data "C:\work\temp\api-refs\V1\sdkfornet-api-ref_v1\Index.html"`

     Untuk `--user-data-dir` argumennya, gunakan nama folder yang berfungsi untuk lingkungan Anda. Folder tidak harus ada.
   - Mulai di: `C:\work\temp\api-refs\V1`
6. Berikan pintasan nama yang sesuai.
7. Buka pintasan untuk melihat referensi API lama.

Lihat Referensi API V2 yang sudah usang

1. Unduh file ZIP untuk Referensi AWS SDK untuk .NET API V2 yang tidak digunakan lagi. Secara default, nama file ZIP adalah`sdkfornet-api-ref_v2_deprecated.zip`. Nama itu akan digunakan di seluruh instruksi ini.
2. Tempatkan file ZIP di folder pilihan Anda. Untuk instruksi ini, nama folder diasumsikan`C:\work\temp\api-refs\V2`.
3. Klik kanan pada file ZIP dan pilih Extract All. Terima lokasi default, yang `C:\work\temp\api-refs\V2\sdkfornet-api-ref_v2` untuk instruksi ini.
4. Buat pintasan untuk Google Chrome di `C:\work\temp\api-refs\V2` folder. Berhati-hatilah untuk tidak memindahkan aplikasi asli.
5. Di properti pintasan baru, atur bidang berikut:

   - Target: "*<path to the Google Chrome application>*" `--disable-web-security --user-data-dir=C:\work\temp\api-refs\V2\data "C:\work\temp\api-refs\V2\sdkfornet-api-ref_v2\Index.html"`

     Untuk `--user-data-dir` argumennya, gunakan nama folder yang berfungsi untuk lingkungan Anda. Folder tidak harus ada.
   - Mulai di: `C:\work\temp\api-refs\V2`
6. Berikan pintasan nama yang sesuai.

7. Buka pintasan untuk melihat referensi API lama.

Riwayat dokumen

Tabel berikut menjelaskan perubahan penting sejak rilis terakhir Panduan AWS SDK untuk .NET Pengembang. Untuk notifikasi tentang pembaruan dokumentasi ini, Anda dapat berlangganan ke umpan RSS.

Perubahan	Deskripsi	Tanggal
Apa yang baru	Menambahkan catatan tentang versi 2.0 dari Alat AWS Deploy untuk.NET CLI.	Oktober 17, 2025
AWS OpsWorks	AWS OpsWorks telah mencapai akhir kehidupan. Itu telah dihapus dari dokumenta si.	September 16, 2025
Apa yang baru	End-of-support telah diumumkan untuk V3 dari. AWS SDK untuk .NET	Agustus 21, 2025
Apa yang baru	Ketersediaan umum untuk Penyedia Cache Terdistribusi	Juli 2, 2025
Kerangka Pemrosesan Pesan	Kerangka Pemrosesan AWS Pesan untuk.NET telah mencapai ketersediaan umum!	5 Mei 2025
Apa yang baru	Versi 4 dari AWS SDK untuk .NET umumnya tersedia! Untuk informasi tentang memigrasi aplikasi ke V4, lihat Memigrasi ke versi 4.	April 28, 2025
Migrasi ke versi 4 dari AWS SDK untuk .NET	Termasuk informasi pratinjau tentang migrasi aplikasi yang dibangun dengan versi AWS SDK untuk .NET ke 4.	Maret 3, 2025

Integrasi AWS dengan .NET Aspire di AWS SDK untuk .NET	Termasuk informasi tentang bagaimana AWS telah Terintegrasi dengan.NET Aspire di AWS SDK untuk .NET.	Februari 15, 2025
Observabilitas	Mengumumkan rilis GA untuk observabilitas	Februari 10, 2025
Apa yang baru	Menambahkan informasi tentang perilaku default baru untuk perlindungan integritas.	Januari 15, 2025
Migrasi V4 Klien Enkripsi S3	Menambahkan informasi tentang migrasi dari V2 ke V4.	Desember 9, 2024
Apa yang baru	Menambahkan informasi tentang rilis pratinjau keempat dari AWS SDK untuk .NET versi 4.	November 15, 2024