デベロッパーガイド

AWS SDK for .NET (V4)

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

AWS SDK for .NET (V4): デベロッパーガイド

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

Amazon の商標およびトレードドレスは、Amazon のものではない製品またはサービスにも関連して、お客様に混乱を招いたり Amazon の信用を傷つけたり失わせたりするいかなる形においても使用することはできません。Amazon が所有していないその他のすべての商標は Amazon との提携、関連、支援関係の有無にかかわらず、それら該当する所有者の資産です。

Table of Contents

..... x
とは AWS SDK for .NET 1
このバージョンについて 1
SDK メジャーバージョンのメンテナンスとサポート 2
一般的なユースケース 2
このセクションのその他のトピック 2
関連 AWS ツール 3
Tools for Windows PowerShell と Tools for PowerShell Core 3
Toolkit for VS Code 3
Toolkit for Visual Studio 3
Toolkit for Azure DevOps 4
サポートされているプラットフォーム 4
.NET Core 4
.NET Standard 2.0 4
.NET Framework 4.5 5
.NET Framework 3.5 5
ポータブルクラスライブラリと Xamarin 5
Unity のサポート 5
SDK とツールのリファレンス 5
改訂履歴 5
最新情報 6
追加リソース 8
入門 10
ツールチェーンのインストールと設定 10
クロスプラットフォーム開発 10
Visual Studio と .NET Core を使用する Windows 環境 11
次のステップ 11
による認証 AWS 11
コンソール認証情報を使用してログインする 12
IAM Identity Center の有効化と設定 13
IAM Identity Center を使用するように SDK を構成します。 13
AWS アクセスポータルセッションを開始する 14
追加情報 15
シンプルなアプリケーションの作成 15

シンプルなクロスプラットフォームアプリ ... 16
シンプルな Windows ベースのアプリ ... 22
次のステップ ... 28
設定 ... 29
新しいプロジェクトを開始する ... 29
NuGet を使用して AWSSDK パッケージをインストールする ... 31
コマンドプロンプトまたはターミナルからの NuGet の使用 ... 32
Visual Studio ソリューションエクスプローラーからの NuGet の使用 ... 32
パッケージマネージャーコンソールからの NuGet の使用 ... 33
NuGet を使用せずに AWSSDK アセンブリをインストールする ... 34
AWS リージョン ... 35
特定のリージョンを使用してサービスクライアントを作成する ... 36
すべてのサービスクライアント用にリージョンを指定する ... 37
リージョン解像度 ... 38
中国 (北京) リージョンに関する特別な情報 ... 38
新しい AWS サービスに関する特別な情報 ... 38
認証情報とプロファイルの解決 ... 39
プロファイルの解決 ... 40
フェデレーティッドユーザーアカウントの認証情報の使用 ... 40
ロールまたは一時認証情報の指定 ... 41
プロキシ認証情報の使用 ... 41
再試行とタイムアウト ... 42
再試行 ... 42
タイムアウト ... 44
オブザーバビリティ ... 46
その他のリソース ... 47
TelemetryProvider の設定 ... 47
メトリクス ... 49
テレメトリプロバイダー ... 50
ユーザーとロール ... 52
ユーザーとアクセス許可セット ... 52
サービス役割 ... 52
高度な設定 ... 53
AWSSDK.Extensions.NETCore.Setup および IConfiguration ... 54
他のアプリケーションパラメータの設定 ... 58
の設定ファイルリファレンス AWS SDK for .NET ... 66

レガシー認証情報の使用 79
認証情報に関する重要な警告とガイダンス 79
共有 AWS 認証情報ファイルの使用 80
SDK ストアの使用 (Windows のみ) 84
SDK の使用 88
非同期プログラミング 88
Pagination (ページ分割) 90
ページネーターの使用の可否 90
ページネーターを使用するメリット 90
同期と非同期のページ分割 91
例 91
ページネーターに関する追加の考慮事項 95
HTTP 2 のサポート 95
追加の考慮事項 98
その他のツール 98
AWS デプロイツール 98
AWS .NET のメッセージ処理フレームワーク 98
.NET Aspire との統合 99
高度な認証 100
シングルサインオン 100
前提条件 101
SSO プロファイルのセットアップ 101
SSO トークンの生成と使用 103
追加リソース 107
チュートリアル 108
チュートリアル: .NET アプリケーションのみ 108
チュートリアル: AWS CLI および .NET アプリケーション 117
にデプロイする AWS 126
.NET CLI からデプロイする 126
IDE ツールキットからデプロイする 126
ユースケース 127
ASP.NET Core Apps 127
.NET コンソールアプリ 128
Blazor WebAssembly アプリ 129
AWS Lambda プロジェクト 130
前提条件 130

使用できる Lambda コマンド 130
デプロイ手順 131
AWS サービスの呼び出し 133
ガイド付きコードの例 133
CloudFormation 134
Amazon Cognito 138
DynamoDB 146
Amazon EC2 176
IAM 238
Amazon S3 258
Amazon SNS 268
Amazon SQS 272
AWS Lambda 305
API 305
前提条件 305
追加情報 306
トピック 306
Lambda Annotations 306
高レベルライブラリとフレームワーク 308
Message Processing Framework 308
.NET Aspire AWS との統合 328
他のサービスと設定 330
コードの例 331
Aurora 332
はじめに 332
基本 334
アクション 358
Auto Scaling 374
はじめに 332
基本 334
アクション 358
Amazon Bedrock 409
はじめに 332
アクション 358
Amazon Bedrock ランタイム 413
Anthropic Claude 414

Cohere Command 419
Meta Llama 423
Mistral AI 427
CloudFormation 431
はじめに 332
CloudWatch 434
はじめに 332
基本 334
アクション 358
CloudWatch Logs 493
アクション 358
シナリオ 496
Amazon Cognito ID プロバイダー 518
アクション 358
AWS Control Tower 519
はじめに 332
基本 334
アクション 358
DynamoDB 559
はじめに 332
基本 334
アクション 358
Amazon EC2 599
はじめに 332
Amazon ECS 601
はじめに 332
AWS IoT 604
はじめに 332
基本 334
アクション 358
AWS IoT data 663
アクション 358
Amazon Redshift 666
はじめに 332
基本 334
アクション 358

Amazon S3 697
開始方法 332
基本 334
アクション 358
シナリオ 496
プロジェクトを移行する 734
バージョン 4 への移行 734
特定のランタイムライブラリまたは .NET Framework の最小バージョンが必要です。 734
値のタイプ 735
コレクション 735
AWS Security Token Service (STS) 736
に関連する変更 ClientConfig 736
AWSSDK.Extensions.NETCore.Setup NuGet パッケージ 737
CookieSigner および UrlSigner 738
DateTime と UTC DateTime 738
DateTime 解析 738
ConvertFromUnixEpochSeconds および ConvertFromUnixEpochMilliseconds 739
ログ記録 739
HTTP 2 のサポート 740
シングルサインオン 740
DynamoDB に固有の変更 740
EC2 に固有の変更 744
S3 に固有の変更 745
削除された要素のプログラミング 748
.NET Standard 1.3 からの移行 751
セキュリティ 753
データ保護 753
Identity and Access Management 755
オーディエンス 755
アイデンティティを使用した認証 756
ポリシーを使用したアクセスの管理 757
IAM AWS のサービス の操作方法 759
AWS ID とアクセスのトラブルシューティング 759
コンプライアンス検証 761
耐障害性 762
インフラストラクチャセキュリティ 762

最小 TLS バージョンの適用 763
.NET Core 763
特定のランタイムライブラリまたは .NET Framework の最小バージョンが必要です。 764
AWS Tools for PowerShell 765
Xamarin 766
Unity 767
ブラウザ (Blazor WebAssembly 用) 767
S3 暗号化クライアントの移行 (V1 から V2) 767
移行の概要 768
既存のクライアントを V1 移行用クライアントに更新して、新しい形式を読み取るようにする 769
V1 移行用クライアントを V2 クライアントに移行して、新しい形式を書き込むようにする 769
V2 クライアントを更新して V1 形式を読み取らないようにする 773
S3 暗号化クライアントの移行 (V2 から V4) 773
V4 の概念について 774
既存のクライアントを更新して V4 形式を読み取る 775
V4 クライアントへの移行 776
その他の設定例 780
特別な考慮事項 783
AWSSDK アセンブリを入手する 783
ZIP ファイルをダウンロードして抽出する 783
アプリケーションでの認証情報とプロファイルへのアクセス 784
クラス CredentialProfileStoreChain の例 785
クラス SharedCredentialsFile と AWSCredentialsFactory の例 786
Unity のサポート 787
Xamarin のサポート 788
API リファレンス 789
API リファレンスバージョンについて 789
ドキュメント履歴 792

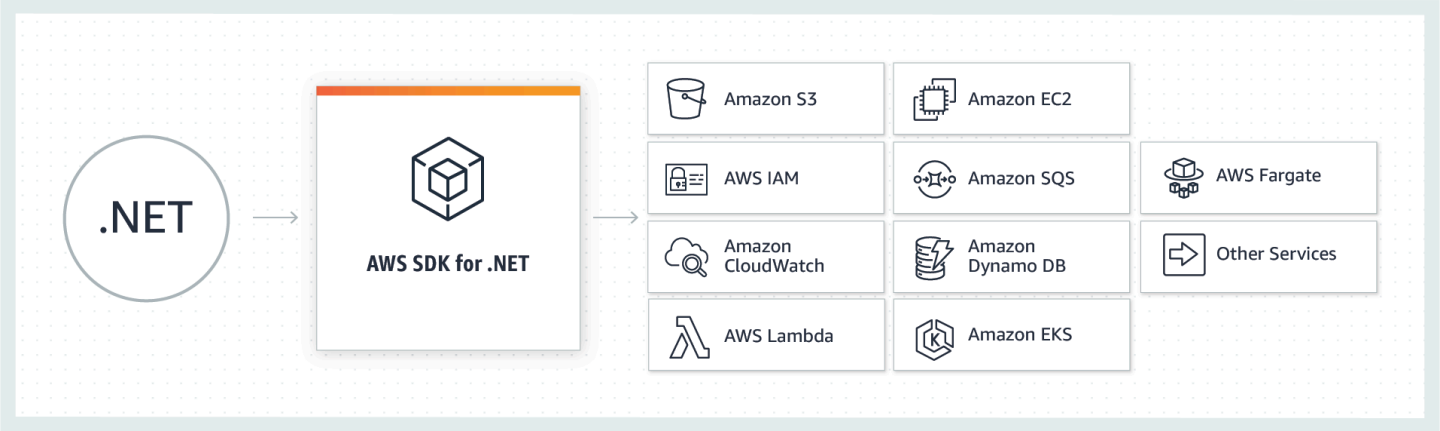
のバージョン 4 (V4) AWS SDK for .NET がリリースされました。

重要な変更とアプリケーションの移行については、「移行トピック」を参照してください。

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。

とは AWS SDK for .NET

AWS SDK for .NET を使用すると、Amazon Simple Storage Service (Amazon S3) や Amazon Elastic Compute Cloud (Amazon EC2) などのコスト効率が高く、スケーラブルで信頼性の高い AWS サービスを利用する .NET アプリケーションを簡単に構築できます。SDK は、.NET 開発者にとって一貫性のある使い慣れた一連のライブラリを提供することで、 AWS サービスの使用を簡素化します。



(OK、取得しました。 セットアップしてクイックツアーに参加する準備が整いました。)

このバージョンについて

現在および過去のドキュメント

このドキュメントは、 のバージョン 4.0 以降 (V4) を対象としています AWS SDK for .NET。主に.NET Core と ASP.NET Core に焦点を当てていますが、.NET Framework および ASP.NET 4.x に関する情報についても説明します。Windows と Visual Studio に加えて、クロスプラットフォーム開発についても同等の考慮事項を提供します。
SDK の以前のバージョンからの移行については、「」を参照してくださいプロジェクトを移行する。
今後廃止される SDK、V3 の以前のバージョンについては、AWS SDK for .NET 「 (V3) デベロッパーガイド」を参照してください。
SDK の以前のバージョンで廃止されたコンテンツを見つけるには、次の項目 (複数可) を参照してください。

- AWS SDK for .NET (バージョン 2、廃止) デベロッパーガイド

- の廃止された API リファレンス AWS SDK for .NET

SDK メジャーバージョンのメンテナンスとサポート

SDK メジャーバージョンのメンテナンスとサポート、およびその基礎的な依存関係については、AWS SDK とツール共有設定および認証情報リファレンスガイドで以下を参照してください。

- AWS SDKsメンテナンスポリシー
- AWS SDKsとツールのバージョンサポートマトリックス

一般的なユースケース

は、次のような魅力的なユースケースを実現する AWS SDK for .NET のに役立ちます。

- AWS Identity and Access Management (IAM) を使用してユーザーおよびロールの管理します。
- Amazon Simple Storage Service (Amazon S3) にアクセスしてバケットを作成し、オブジェクトを格納します。
- トピックの Amazon Simple Notification Service (Amazon SNS) HTTP サブスクリプションを管理します。
- S3 転送ユーティリティを使用して、Xamarin アプリケーションから Amazon S3 にファイルを転送します。
- Amazon Simple Queue Service (Amazon SQS) を使用して、システム内のコンポーネント間のメッセージとワークフローを処理します。
- SQL ステートメントを Amazon S3 Select に送信して、効率的な Amazon S3 転送操作を実行します。
- Amazon EC2 インスタンスを作成および起動して、Amazon EC2 スポットインスタンスを設定およびリクエストします。

このセクションのその他のトピック

- AWS に関連する ツール AWS SDK for .NET
- でサポートされているプラットフォーム AWS SDK for .NET

- AWS SDKsとツールのリファレンスガイド
- 改訂履歴
- の新機能 AWS SDK for .NET
- 追加リソース

AWS に関連する ツール AWS SDK for .NET

Tools for Windows PowerShell と Tools for PowerShell Core

AWS Tools for Windows PowerShell および AWS Tools for PowerShell Core は、 によって公開される機能上に構築された PowerShell モジュールです AWS SDK for .NET。 AWS PowerShell ツールを使用すると、PowerShell プロンプトから AWS リソースに対するオペレーションをスクリプト化できます。コマンドレットの実装には SDK のサービスクライアントとメソッドが使用されていますが、コマンドでは PowerShell の慣用的な方法に従ってパラメータを指定し、結果を処理できます。

開始するには、「AWS Tools for Windows PowerShell」を参照してください。

Toolkit for VS Code

AWS Toolkit for Visual Studio Code は、Visual Studio Code (VS Code) エディタ用のプラグインです。このツールキットを使用すると、 AWSを使用するアプリケーションの開発、デバッグ、およびデプロイが容易になります。

このツールキットでは、次のような操作ができます。

- AWS Lambda 関数を含むサーバーレスアプリケーションを作成し、アプリケーションを AWS CloudFormation スタックにデプロイします。
- Amazon EventBridge スキーマを使用する。
- IntelliSense を使用して Amazon ECS タスク定義ファイルを操作する。
- AWS Cloud Development Kit (AWS CDK) アプリケーションを視覚化します。

Toolkit for Visual Studio

AWS Toolkit for Visual Studio は、Amazon Web Services を使用する .NET アプリケーションの開発、デバッグ、デプロイを容易にする Visual Studio IDE 用のプラグインです。Toolkit for Visual Studio は、Lambda などのサービス用の Visual Studio テンプレートと、ウェブアプリケーション

やサーバーレスアプリケーション用のデプロイウィザードを提供します。 AWS Explorer を使用して、Amazon EC2 インスタンスの管理、Amazon DynamoDB テーブルの操作、Amazon Simple Notification Service (Amazon SNS) キューへのメッセージの発行などのすべての操作を Visual Studio 内で実行できます。

開始するには、「AWS Toolkit for Visual Studioのセットアップ」を参照してください。

Toolkit for Azure DevOps

にタスク AWS Toolkit for Microsoft Azure DevOps が追加され、Azure DevOps および Azure DevOps Server のビルドパイプラインとリリースパイプラインが AWS サービスと簡単に連携できるようになります。Amazon S3、 AWS Elastic Beanstalk、 AWS CodeDeploy Lambda、、Amazon Simple Queue Service (Amazon SQS) AWS CloudFormation、および Amazon SNS を使用できます。Windows PowerShell モジュールと AWS Command Line Interface () を使用してコマンドを実行することもできますAWS CLI。

の使用を開始するには AWS Toolkit for Azure DevOps、 AWS Toolkit for Microsoft Azure DevOps ユーザーガイドを参照してください。

でサポートされているプラットフォーム AWS SDK for .NET

AWS SDK for .NET は、開発者がさまざまなプラットフォームをターゲットにするためのアセンブリの個別のグループを提供します。ただし、SDK のすべての機能がそれぞれのプラットフォームで同一というわけではありません。このトピックでは、各プラットフォームでのサポートに違いについて説明します。

.NET Core

は、.NET Core (.NET Core 3.1、.NET 5、.NET 6 など) 用に記述されたアプリケーション AWS SDK for .NET をサポートします。 AWS サービスクライアントは、.NET Core の非同期呼び出しパターンのみをサポートします。これは、.NET Core 環境で非同期呼び出しのみをサポートしている Amazon S3 の `TransferUtility` のようなサービスクライアントの環境で構築された高レベルの抽象化の多くにも影響します。

.NET Standard 2.0

のフレームワーク以外のバリエーションは、.NET Standard 2.0 AWS SDK for .NET に準拠しています。 AWS SDK for .NET は、.NET Standard に対して作成されたアプリケーションに対して非同期メソッドのみを提供します。

.NET Framework 4.5

このバージョンの AWS SDK for .NET は、.NET Framework 4.7.2 に対してコンパイルされ、.NET 4.0 runtime. AWS service クライアントで実行され、同期および非同期呼び出しパターンをサポートし、C# 5.0 で導入された非同期および待機キーワードを使用します。

.NET Framework 3.5

のバージョン 4 AWS SDK for .NET は .NET Framework 3.5 をサポートしていません。

ポータブルクラスライブラリと Xamarin

には、ポータブルクラスライブラリの実装 AWS SDK for .NET も含まれています。ポータブルクラスライブラリの実装では、ユニバーサル Windows プラットフォーム (UWP) や、Android と iOS の Xamarin など、複数のプラットフォームを対象にすることができます。詳細については、Mobile SDK for .NET and Xamarin を参照してください。 AWS サービスクライアントは非同期呼び出しパターンのみをサポートします。

Unity のサポート

Unity のサポートについては、「Unity のサポートに関する特別な考慮事項」を参照してください。

AWS SDKsとツールのリファレンスガイド

AWS SDKsとツールのリファレンスガイドには、多くの AWS SDKs とツールキット、および に関連する重要な情報が含まれています AWS CLI。以下は、レファレンスに含まれる情報のいくつかの例です。

- 共有 AWSconfig ファイルと `credentials` ファイルとその場所に関する情報。
- AWS アカウント、ユーザー、ロールのセットアップ
- 設定と認証設定のリファレンス
- AWS 共通ランタイム (CRT) ライブラリ
- AWS SDKsメンテナンスポリシー
- AWS SDKsとツールのバージョンサポートマトリックス

改訂履歴

さまざまなリリースでの変更点については、以下を参照してください。

- SDK 変更ログ
- の新機能 AWS SDK for .NET
- ドキュメント履歴

の新機能 AWS SDK for .NET

に関連する新しい開発の概要については AWS SDK for .NET、https://aws.amazon.com/sdk-for-net/ の製品ページと SDK 変更ログを参照してください。

AWS SDK for .NETの最新情報は次のとおりです。

2025 年 10 月 17 日: AWS デプロイツールのバージョン 2.0

.NET CLI 用の AWS デプロイツールのバージョン 2.0 がリリースされました。デプロイツールの詳細については、「」を参照してくださいアプリケーションを にデプロイする AWS。Deploy Tool のバージョン 2.0 の詳細については、ブログ記事「Deploy Tool for .NET」の AWS 「最新情報」を参照してください。

2025 年 8 月 21 日: のバージョン 3 end-of-support AWS SDK for .NET

のバージョン 3 (V3) end-of-support AWS SDK for .NET が発表されました。V3 アプリケーションを移行し、中断を回避するには、移行ガイドを参照してください。詳細については、ブログ記事「 AWS SDK for .NET v3 end-of-supportを発表する」を参照してください。

2025 年 7 月 2 日: 分散キャッシュプロバイダーの一般提供開始

Amazon DynamoDB 用の AWS .NET 分散キャッシュプロバイダーは一般利用可能です。このプロバイダーは ASP.NET Core IDistributedCache インターフェイスを実装しているため、コードの変更を最小限に抑えながら、DynamoDB のフルマネージドで耐久性の高いインフラストラクチャをキャッシュレイヤーに統合できます。分散キャッシュプロバイダーは、NuGet の AWS.AspNetCore.DistributedCacheProvider パッケージから入手できます。詳細については、ブログ記事AWS 「.NET Distributed Cache Provider for Amazon DynamoDB now generally Available」を参照してください。

> **Note**
>
> リリースされたバージョンの分散キャッシュプロバイダーは、 の V4 でのみサポートされています AWS SDK for .NET。分散キャッシュプロバイダーのプレビューバージョンを試した場合は、アプリケーションのパッケージの依存関係を更新する必要がある場合があります。

2025 年 5 月 5 日: AWS Message Processing Framework for .NET の一般提供開始

AWS Message Processing Framework for .NET は一般利用可能です。フレームワークは、Amazon Simple Queue Service (SQS)、Amazon Simple Notification Service (SNS)、Amazon EventBridge などの AWS サービスを使用する .NET メッセージ処理アプリケーションの開発を簡素化する AWSネイティブフレームワークです。

2025 年 4 月 28 日: のバージョン 4 AWS SDK for .NET

のバージョン 4 AWS SDK for .NET は一般利用可能です。アプリケーションを V4 に移行する方法については、「」を参照してくださいバージョン 4 への移行。また、一般提供を発表するブログ記事 AWS SDK for .NET V4.0 の一般提供」も参照してください。

2025 年 2 月 15 日: .NET Aspire との統合

内部開発ループを改善するための .NET Aspire との統合がリリースされました。詳細については、「での .NET Aspire AWS との統合 AWS SDK for .NET」を参照してください。

2025 年 2 月 10 日: オブザーバビリティのための GA リリース

オブザーバビリティは、システムが出力するデータからシステムの現在の状態を推測できる範囲です。テレメトリプロバイダーの実装など AWS SDK for .NET、オブザーバビリティが に追加されました。詳細については、このガイドオブザーバビリティの「」およびブログ記事「. AWS NET OpenTelemetry ライブラリの一般提供開始のお知らせ」を参照してください。

2025 年 1 月 15 日: 整合性保護の新しいデフォルト動作

のバージョン 3.7.412.0 以降 AWS SDK for .NET、SDK はアップロードのCRC32チェックサムを自動的に計算することで、デフォルトの整合性保護を提供します。詳細については、 の GitHub のお知らせを参照してくださいhttps://github.com/aws/aws-sdk-net/issues/3610。SDK には、外部で設定できるデータ整合性保護のグローバル設定も用意されています。詳細については、AWS SDKs「データ整合性保護」を参照してください。

2024 年 11 月 15 日: バージョン 4 のプレビュー 4 リリースのブログ投稿

AWS SDK for .NET バージョン 4 のプレビュー 4 は、2024 年 11 月 15 日にリリースされました。このプレビューの詳細については、ブログ記事「Preview 4 of AWS SDK for .NET V4」を参照してください。

2024 年 8 月 16 日: バージョン 4 のプレビュー 1 リリースのブログ投稿

AWS SDK for .NET バージョン 4 は、2024 年 8 月 16 日に最初のプレビューとしてリリースされました。このプレビューの詳細については、ブログ記事「プレビュー 1 of AWS SDK for .NET V4」を参照してください。

追加リソース

サポートされる サービス

はほとんどの AWS インフラストラクチャ製品 AWS SDK for .NET をサポートし、さらに多くのサービスが頻繁に追加されます。SDK でサポートされている AWS サービスのリストについては、 SDK README ファイルを参照してください。

のホームページ AWS SDK for .NET

の詳細については AWS SDK for .NET、https://aws.amazon.com/sdk-for-net/ の SDK のホームページを参照してください。

SDK リファレンスドキュメント

SDK リファレンスドキュメントでは、SDK に付属するすべてのコードを参照して検索できます。詳細なドキュメントと使用例を確認できます。詳細については、「 APIリファレンスAWS SDK for .NET」を参照してください。

AWS re:Post (以前の AWS フォーラム)

AWS re:Post、特に のトピック AWS SDK for .NETにアクセスして、質問したりフィードバックを提供したりできます AWS。各ドキュメント ページの下部には、関連する re:Post トピックに移動できる「Try AWS re:Post」リンクがあります。 AWS エンジニアはトピックを監視し、質問、フィードバック、問題に対応します。

re: POST にログインしていれば、トピックをフォローすることもできます。のトピックに従うには AWS SDK for .NET、「すべてのトピック」ページに移動し、「.NET on AWS」を見つけて、フォローボタンを選択します。

ツールキット

- AWS Toolkit for Visual Studio：Microsoft Visual Studio IDE を使用する場合は、AWS Toolkit for Visual Studio ユーザーガイドを確認いただく必要があります。
- AWS Toolkit for Visual Studio Code：Microsoft Visual Studio IDE を使用する場合は、AWS Toolkit for Visual Studio Code ユーザーガイドを確認いただく必要があります。

便利なライブラリ、拡張機能、およびツール

Github ウェブサイトの aws/dotnet と aws/aws-sdk-net リポジトリには、 AWSで .NET アプリケーションとサービスを構築するために役立つライブラリ、ツール、およびリソースへのリンクが用意されています。

次に例をいくつか示します。

- AWS Systems Manager の .NET Configuration Extension
- AWS 拡張機能 .NET Core セットアップ
- AWS .NET のログ記録
- Amazon Cognito Authentication 拡張ライブラリ
- AWS X-Ray SDK for .NET

その他のリソース

役に立つと思われるその他のリソースは次のとおりです。

- デベロッパーネット
- AWS クラウド上の .NET 開発環境 - クイックスタートリファレンスデプロイ
- こんにちは、クラウド！ブログ
- AWS ホワイトペーパー: での .NET アプリケーションの開発とデプロイ AWS
- AWS Microservice Extractor for .NET
- Porting Assistant for .NET
- AWS SDKsとツールのリファレンスガイド

の開始方法 AWS SDK for .NET

をインストール、セットアップ、使用する方法について説明します AWS SDK for .NET。ツールチェーンをインストールし、アプリケーションが AWS サービスにアクセスするために必要ないくつかの重要な事項を設定する必要があります。具体的には次のとおりです。

- 適切なユーザーアカウントまたはロール
- そのユーザーアカウントの、またはそのロールを引き受けるための認証情報

プロジェクトやその他の重要な部分を設定する方法については、「」を参照してくださいの設定 AWS SDK for .NET。

トピック

- 用のツールチェーンのインストールと設定 AWS SDK for .NET
- AWS SDK for .NET を使用した の認証 AWS
- を使用したシンプルなアプリケーションの作成 AWS SDK for .NET

用のツールチェーンのインストールと設定 AWS SDK for .NET

を使用するには AWS SDK for .NET、特定の開発ツールがインストールされている必要があります。

クロスプラットフォーム開発

Windows、Linux、または macOS でのクロスプラットフォームの .NET 開発では以下が必要です。

- Microsoft .NET Core SDK、バージョン 2.1、3.1 以降。.NET コマンドラインインターフェイス (CLI) (**`dotnet`**) および .NET Core ランタイムを含むものとします。

- 使用しているオペレーティングシステムと要件に適したコードエディタまたは統合開発環境 (IDE)。これは通常、.NET Core のサポートを含むものです。

  例として、Microsoft Visual Studio Code (VS Code)、JetBrains Rider、Microsoft Visual Studio があります。

- (オプション) 選択したエディタとオペレーティングシステムで AWS 使用可能なツールキット。

例として AWS Toolkit for Visual Studio Code、AWS Toolkit for JetBrains、AWS Toolkit for Visual Studio があります。

Visual Studio と .NET Core を使用する Windows 環境

Visual Studio および .NET Core を使用した Windows での開発では以下が必要です。

- Microsoft Visual Studio

- Microsoft .NET Core 2.1, 3.1 以降

  これは通常、Visual Studio の最新バージョンをインストールするとデフォルトで追加されます。

- (オプション) は AWS Toolkit for Visual Studio、Visual Studio から AWS リソースとローカルプロファイルを管理するためのユーザーインターフェイスを提供するプラグインです。ツールキットをインストールするには、「AWS Toolkit for Visual Studioのセットアップ」を参照してください。

  詳細については、「AWS Toolkit for Visual Studio ユーザーガイド」を参照してください。

次のステップ

AWS SDK for .NET を使用した の認証 AWS

AWS SDK for .NET を使用した の認証 AWS

開発 AWS 時にコードが で認証される方法を確立する必要があります AWS のサービス。環境と利用可能なアクセスに応じて、 AWS リソースへのプログラムによる AWS アクセスを設定する方法はさまざまです。

SDK のさまざまな認証方法を確認するには、「AWS SDK およびツール リファレンス ガイド」の「認証とアクセス」を参照してください。

このトピックでは、新しいユーザーがローカルで開発中であり、雇用主から認証方法が与えられておらず、 AWS IAM アイデンティティセンター を使用して一時的な認証情報を取得することを前提としています。ご使用の環境がこれらの前提条件に当てはまらない場合、このトピックの情報の一部はお客様に該当しない場合や、既に提供されている可能性があります。

この環境を構成するにはいくつかのステップが必要で、その概要は以下のとおりです。

1. コンソール認証情報を使用してログインする
2. IAM Identity Center の有効化と設定
3. IAM Identity Center を使用するように SDK を構成します。
4. AWS アクセスポータルセッションを開始する

コンソール認証情報を使用してログインする

既存の AWS マネジメントコンソールのサインイン認証情報を使用して、 AWS サービスにプログラムでアクセスできます。ブラウザベースの認証フローの後、 は CLI AWS Tools for PowerShell や AWS SDK for AWS .NET v4 などのローカル開発ツールで動作する一時的な認証情報 AWS を生成します。

このプロセスでは、最初のアカウント設定時に作成されたルート認証情報、IAM ユーザー、または ID プロバイダーからのフェデレーティッド ID を使用して認証でき、CLI AWS または AWS Tools for PowerShell が自動的に一時的な認証情報を管理します。このアプローチにより、長期的な認証情報をローカルに保存する必要がなくなるため、セキュリティが強化されます。コマンド`Invoke-AWSLogin` レットを から AWS Tools for PowerShell 実行するか、CLI AWS から `aws login` コマンドを実行すると、アクティブなコンソールセッションから選択するか、ブラウザベースの認証フローを使用してサインインできます。これにより、一時的な認証情報が自動的に生成されます。一時的な認証情報トークンは 15 分で期限切れになりますが、 AWS SDK for .NET v4 はリクエスト中に必要に応じてトークンを自動的に更新します。

> **Important**
>
> 認証にコンソール認証情報を使用している場合、 AWS サインイン解決が機能するように、アプリケーションは次の NuGet パッケージを参照する必要があります。
>
> - AWSSDK.Signin
>
> これらのパッケージを参照しないと、ランタイム 例外が発生します。

- AWS Tools for PowerShell login creds デベロッパーガイド。
- CLI を使用してコンソール認証情報を使用して AWS ローカル開発にログイン AWS します。

AWS Tools for PowerShell または AWS CLI の上記の手順に従ってログインすると、configファイルは次のようになります。

```
[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
```

IAM Identity Center の有効化と設定

IAM Identity Center を使用するには、まず IAM Identity Center を有効にして構成する必要があります。SDK でこれを行う方法の詳細については、「AWS SDK およびツール リファレンス ガイド」の IAM Identity Center 認証に関するトピックのステップ 1を参照してください。具体的には、「IAM Identity Center 経由のアクセスを確立していません」にある必要な指示に従ってください。

IAM Identity Center を使用するように SDK を構成します。

IAM Identity Center を使用するように SDK を設定する方法に関する情報は、「AWS SDK およびツールリファレンスガイド」の IAM Identity Center 認証に関するトピックのステップ 2 に記載されています。この設定を完了すると、システムには以下の要素が含まれるようになります。

- アプリケーションを実行する前に AWS アクセスポータルセッションを開始 AWS CLIするために使用する 。
- SDK から参照できる設定値のセットを持つ[default]プロファイルを含む共有 AWS configファイル。このファイルの場所を確認するには、「AWS SDK とツールのリファレンスガイド」の「共有ファイルの場所」を参照してください。 AWS SDK for .NET は、リクエストを送信する前に、プロファイルの SSO トークンプロバイダーを使用して認証情報を取得しますAWS。IAM Identity Center 許可セットに接続された IAM ロールである sso_role_name 値により、アプリケーションで使用されている AWS のサービス にアクセスできます。

  次のサンプル config ファイルは、SSO トークンプロバイダーで設定されたデフォルトプロファイルを示しています。プロファイルの sso_session 設定は、指定された sso-session セクションを参照します。sso-session セクションには、 AWS アクセスポータルセッションを開始するための設定が含まれています。

  ```
  [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**
>
> 認証 AWS IAM アイデンティティセンター に を使用している場合、アプリケーションは SSO 解決が機能するように、次の NuGet パッケージを参照する必要があります。
>
> - `AWSSDK.SSO`
> - `AWSSDK.SSOOIDC`
>
> これらのパッケージを参照しないと、ランタイム 例外が発生します。

AWS アクセスポータルセッションを開始する

アクセスするアプリケーションを実行する前に AWS のサービス、SDK が IAM Identity Center 認証を使用して認証情報を解決するためのアクティブな AWS アクセスポータルセッションが必要です。設定したセッションの長さによっては、アクセスが最終的に期限切れになり、SDK で認証エラーが発生します。 AWS アクセスポータルにサインインするには、 で次のコマンドを実行します AWS CLI。

```
aws sso login
```

デフォルトのプロファイルを設定している場合は、`--profile` オプションを指定してコマンドを呼び出す必要はありません。SSO トークンプロバイダー設定で名前付きプロファイルを使用している場合、コマンドは `aws sso login --profile named-profile` です。

アクティブなセッションが既にあるかどうかをテストするには、次の AWS CLI コマンドを実行します。

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

このコマンドへの応答により、共有 config ファイルに設定されている IAM Identity Center アカウントとアクセス許可のセットが報告されます。

> **Note**
>
> 既にアクティブな AWS アクセスポータルセッションがあり、 を実行している場合はaws sso login、認証情報を指定する必要はありません。
> サインインプロセスにより、データ AWS CLI へのアクセスを許可するように求められる場合があります。 AWS CLI は SDK for Python 上に構築されているため、アクセス許可メッセージにはbotocore名前のバリエーションが含まれている可能性があります。

追加情報

- 開発環境での IAM Identity Center と SSO の使用に関する追加情報については、高度な認証セクションのシングルサインオンを参照してください。この情報には、代替方法やより高度な方法、そしてこれらの方法の使い方を紹介するチュートリアルが含まれています。
- プロファイルや環境変数の使用など、SDK の認証に関するその他のオプションについては、「AWS SDK およびツールリファレンスガイド」の設定に関する章を参照してください。
- ベストプラクティスの詳細については、IAM ユーザーガイドの「IAM でのセキュリティのベストプラクティス」を参照してください。
- 短期 AWS 認証情報を作成するには、IAM ユーザーガイドの「一時的なセキュリティ認証情報」を参照してください。
- その他の認証情報プロバイダーについては、「AWS SDK とツールのリファレンスガイド」の「標準化された認証情報プロバイダー」を参照してください。

を使用したシンプルなアプリケーションの作成 AWS SDK for .NET

このセクションでは、 AWS SDK for .NETを初めて使用する開発者向けの基本的なチュートリアルを提供します。

> **Note**
>
> これらのチュートリアルを使用する前に、まずツールチェーンをインストールし、SDK 認証を設定しておく必要があります。

特定の AWS サービス用のソフトウェアの開発とコード例については、「」を参照してください AWS サービスの呼び出し。その他のコード例については、「SDK for .NET (v4) コード例」を参照してください。

トピック

- を使用したシンプルなクロスプラットフォームアプリケーション AWS SDK for .NET
- を使用したシンプルな Windows ベースのアプリケーション AWS SDK for .NET
- 次のステップ

を使用したシンプルなクロスプラットフォームアプリケーション AWS SDK for .NET

このチュートリアルでは、クロスプラットフォーム開発に AWS SDK for .NET と .NET Core を使用します。また、SDK を使用して、所有する Amazon S3 バケットを一覧表示し、必要に応じてバケットを作成する方法を説明します。

このチュートリアルは、.NET コマンドラインインターフェイス (CLI) などのクロスプラットフォームツールを使用して実行します。開発環境を設定するその他の方法については、「用のツールチェーンのインストールと設定 AWS SDK for .NET」を参照してください。

Windows、Linux、または macOS でのクロスプラットフォームの .NET 開発では必須:

- Microsoft .NET Core SDK、バージョン 2.1、3.1 以降。.NET コマンドラインインターフェイス (CLI) (**`dotnet`**) および .NET Core ランタイムを含むものとします。

- 使用しているオペレーティングシステムと要件に適したコードエディタまたは統合開発環境 (IDE)。これは通常、.NET Core のサポートを含むものです。

  例として、Microsoft Visual Studio Code (VS Code)、JetBrains Rider、Microsoft Visual Studio があります。

> **Note**
>
> これらのチュートリアルを使用する前に、まずツールチェーンをインストールし、SDK 認証を設定しておく必要があります。

ステップ

- プロジェクトの作成
- コードの作成
- アプリケーションを実行する
- クリーンアップ

プロジェクトの作成

1. コマンドプロントまたはターミナルを開きます。.NET プロジェクトを作成できるオペレーティングシステムフォルダを検索するか作成します。
2. そのフォルダで、次のコマンドを実行して .NET プロジェクトを作成します。

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

3. 新しく作成した `S3CreateAndList` フォルダに移動し、次のコマンドを実行します。

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

上記のコマンドは、NuGet パッケージ マネージャーから NuGet パッケージをインストールします。このチュートリアルに必要な NuGet パッケージは正確にわかっているため、ここでこのステップを実行できます。また、開発中に必要なパッケージが判明することは一般的なことです。その場合は、その時点で同様のコマンドを実行できます。

コードの作成

1. `S3CreateAndList` フォルダで、`Program.cs` を見つけてコードエディタで開きます。

2. 内容を次のコードに置き換えて、ファイルを保存します。

```
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;
        }
    }
}
```

アプリケーションを実行する

1. 以下のコマンドを実行してください。

```
dotnet run
```

2. 出力を調べ、所有している Amazon S3 バケットの数 (存在する場合) とその名前を確認します。
3. 新しい Amazon S3 バケットの名前を選択します。「dotnet-quicktour-s3-1-cross-」をベースとして使用し、GUID や名前などのような一意のものを追加します。「Amazon S3 ユーザーガイド」の「バケット命名規則」で説明されているように、バケット名のルールに従ってください。
4. 次のコマンドを実行し、*amzn-s3-demo-bucket* を選択したバケットの名前に置き換えます。

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

5. 出力を調べて、作成された新しいバケットを確認します。

クリーンアップ

このチュートリアルでは、この時点でクリーンアップを選択できるいくつかのリソースを作成しました。

- 前のステップでアプリケーションが作成したバケットを保持しない場合は、https://console.aws.amazon.com/s3/ の Amazon S3 コンソールを使用してバケットを削除します。
- .NET プロジェクトを保持しない場合は、開発環境から `S3CreateAndList` フォルダを削除します。

次の段階

クイックツアーメニューに戻るか、このクイックツアーの最後までスキップします。

を使用したシンプルな Windows ベースのアプリケーション AWS SDK for .NET

このチュートリアルでは、Visual Studio と .NET Core を備えた Windows AWS SDK for .NET 上の を使用します。また、SDK を使用して、所有する Amazon S3 バケットを一覧表示し、必要に応じてバケットを作成する方法を説明します。

このチュートリアルは、Visual Studio と .NET Core を使用して Windows で実行します。開発環境を設定するその他の方法については、「用のツールチェーンのインストールと設定 AWS SDK for .NET」を参照してください。

Visual Studio および .NET Core を使用した Windows での開発では必須:

- Microsoft Visual Studio
- Microsoft .NET Core 2.1, 3.1 以降

  これは通常、Visual Studio の最新バージョンをインストールするとデフォルトで追加されます。

> **Note**
>
> これらのチュートリアルを使用する前に、まずツールチェーンをインストールし、SDK 認証を設定しておく必要があります。

ステップ

- プロジェクトの作成
- コードの作成
- アプリケーションを実行する
- クリーンアップ

プロジェクトの作成

1. Visual Studio を開き、コンソールアプリテンプレートの C# バージョンを使用する新しいプロジェクトを作成します。つまり、「....NET で実行できるコマンドラインアプリケーションを作

成するためのもの」と記述します。プロジェクトに S3CreateAndList という名前を付けます。

> **Note**
>
> コンソールアプリケーションテンプレートの .NET Framework バージョンを選択しないでください。選択する場合は、.NET Framework 4.7.2 以降を使用してください。

2. 新しく作成したプロジェクトを読み込んだら、[ツール]、[NuGet パッケージマネージャー]、[ソリューションの NuGet パッケージの管理] の順に選択します。
3. 次の NuGet パッケージをブラウズし、プロジェクト AWSSDK.S3、AWSSDK.SecurityToken、AWSSDK.SSO、AWSSDK.SSOOIDC にインストールします。

   このプロセスでは、NuGet パッケージ マネージャーから NuGet パッケージをインストールします。このチュートリアルに必要な NuGet パッケージは正確にわかっているため、ここでこのステップを実行できます。また、開発中に必要なパッケージが判明することは一般的なことです。その場合は、その時点で同様の手順に従ってインストールしてください。
4. コマンドプロンプトからアプリケーションを実行する場合は、ここでコマンドプロンプトを開き、ビルド出力を含むフォルダに移動します。通常は S3CreateAndList\S3CreateAndList\bin\Debug\net6.0 のようになりますが、環境によって異なります。

コードの作成

1. S3CreateAndList プロジェクトで、Program.cs を探して IDE で開きます。
2. 内容を次のコードに置き換えて、ファイルを保存します。

```
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. アプリケーションをビルドします。

> **Note**
>
> 古いバージョンの Visual Studio を使用している場合は、次のようなビルドエラーが発生することがあります。
> "Feature 'async main' is not available in C# 7.0. Please use language version 7.1 or greater." (C# 7.0 では async main 機能を使用できません。言語バージョン 7.1 以降を使用してください。)

このエラーが発生した場合は、新しいバージョンの言語を使用するようにプロジェクトをセットアップします。これは通常、プロジェクトのプロパティ (Build、Advanced) で実行します。

アプリケーションを実行する

1. コマンドライン引数なしでアプリケーションを実行します。これは、コマンドプロンプト (すでに開いている場合) または IDE から行います。
2. 出力を調べ、所有している Amazon S3 バケットの数 (存在する場合) とその名前を確認します。
3. 新しい Amazon S3 バケットの名前を選択します。「dotnet-quicktour-s3-1-winvs-」をベースとして使用し、GUID や名前などのような一意のものを追加します。「Amazon S3 ユーザーガイド」の「バケット命名規則」で説明されているように、バケット名のルールに従ってください。
4. アプリケーションを再度実行します。今回はバケット名を指定します。

   コマンドラインで、次のコマンドの *amzn-s3-demo-bucket* を、選択したバケットの名前に置き換えます。

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

   または、IDE でアプリケーションを実行している場合は、[プロジェクト]、[S3CreateAndList プロパティ]、[デバッグ] の順に選択し、そこにバケット名を入力します。
5. 出力を調べて、作成された新しいバケットを確認します。

クリーンアップ

このチュートリアルでは、この時点でクリーンアップを選択できるいくつかのリソースを作成しました。

- 前のステップでアプリケーションが作成したバケットを保持しない場合は、https://console.aws.amazon.com/s3/ の Amazon S3 コンソールを使用してバケットを削除します。
- .NET プロジェクトを保持しない場合は、開発環境から `S3CreateAndList` フォルダを削除します。

次の段階

クイックツアーメニューに戻るか、このクイックツアーの最後までスキップします。

次のステップ

これらのチュートリアルの実行中に作成した残りのリソースは必ずすべてクリーンアップしてください。これらは、ファイルやフォルダなどの開発環境の AWS リソースまたはリソースである場合があります。

を学習したので AWS SDK for .NET、プロジェクトを開始できます。

の設定 AWS SDK for .NET

を使用するには AWS SDK for .NET、環境をセットアップし、アプリケーションが AWS サービスにアクセスするために必要ないくつかの重要なものを設定する必要があります。具体的には次のとおりです。

- AWS リージョンの仕様
- AWSSDK パッケージまたはアセンブリ

このセクションのいくつかのトピックでは、これらの重要な項目の設定方法について説明します。

このセクションの他のトピックおよび他のセクションでは、プロジェクトを設定するさらに高度な方法について説明します。

トピック

- 新しいプロジェクトを開始する
- NuGet を使用して AWSSDK パッケージをインストールする
- NuGet を使用せずに AWSSDK アセンブリをインストールする
- の AWS リージョンの設定 AWS SDK for .NET
- 認証情報とプロファイルの解決
- 再試行とタイムアウト
- オブザーバビリティ
- ユーザーとロールに関する追加情報
- AWS SDK for .NET プロジェクトの高度な設定
- レガシー認証情報の使用

新しいプロジェクトを開始する

AWS サービスにアクセスするための新しいプロジェクトを開始するために使用できるテクニックがいくつかあります。これらのテクニックの一部を次に示します。

- で .NET 開発を初めて使用する場合、 AWS または少なくとも を初めて使用する場合は AWS SDK for .NET、「」で完全な例を確認できますシンプルなアプリケーションの作成。SDK についてわかりやすく説明しています。

- 基本的なプロジェクトは、.NET CLI を使用して開始できます。この例を表示するには、コマンドプロンプトまたはターミナルを開き、フォルダまたはディレクトリを作成してそこに移動し、次のように入力します。

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

  空のプロジェクトが作成されます。これにコードと NuGet パッケージを追加できます。詳細については、.NET Core ガイドを参照してください。

  プロジェクトテンプレートのリストを表示するには、以下を使用します。`dotnet new --list`

- AWS Toolkit for Visual Studio には、さまざまな AWS サービス用の C# プロジェクトテンプレートが含まれています。Visual Studio でツールキットをインストールすると、これらのテンプレートにアクセスして新しいプロジェクトを作成できます。

  これを確認するには、「 AWS Toolkit for Visual Studio ユーザーガイド」の「 AWS サービスの使用」を参照してください。このセクションに、新しいプロジェクトの作成例がいくつか含まれています。

- Windows で Visual Studio を使用して開発し、 を使用しない場合は AWS Toolkit for Visual Studio、新しいプロジェクトを作成するための一般的な手法を使用します。

  例を確認するには、Visual Studio を開き、[File] (ファイル)、[New] (新規作成)、[Project] (プロジェクト) の順に選択します。「.net core」を検索し、コンソールアプリ (.NET Core) または WPF アプリ (.NET Core) テンプレートを選択します。空のプロジェクトが作成されます。これにコードと NuGet パッケージを追加できます。

AWS のサービスの操作方法の例は、「」で確認できますガイド付きコードの例。

> **Important**
>
> 認証 AWS IAM アイデンティティセンター に を使用している場合、アプリケーションは SSO 解決が機能するように、次の NuGet パッケージを参照する必要があります。
>
> - `AWSSDK.SSO`
> - `AWSSDK.SSOOIDC`

これらのパッケージを参照しないと、ランタイム例外が発生します。

NuGet を使用して AWSSDK パッケージをインストールする

NuGet は .NET プラットフォームのパッケージ管理システムです。NuGet を使用すると、AWSSDK パッケージとその他のいくつかの拡張機能をプロジェクトにインストールできます。詳細については、GitHub ウェブサイトの aws/dotnet リポジトリを参照してください。

NuGet には常に AWSSDK パッケージの最新バージョンと以前のバージョンが含まれています。NuGet はパッケージ間の依存関係を認識し、必要なすべてのパッケージを自動的にインストールします。

Warning

NuGet パッケージのリストには、単に「AWSSDK」という名前の (識別子が付加されていない) パッケージが含まれていることがあります。この NuGet パッケージはインストールしないでください。これはレガシーであるため、新しいプロジェクトには使用できません。

NuGet を使用してインストールしたパッケージは、一元的な場所ではなく、プロジェクトと一緒の場所に保存されます。これにより、その他のアプリケーションの互換性問題を発生させることなく、特定のアプリケーションに固有のアセンブリバージョンをインストールすることができます。NuGet の詳細については、NuGet のドキュメントを参照してください。

Note

プロジェクトベースで NuGet パッケージをダウンロードしてインストールできない、または許可されていない場合は、AWSSDK アセンブリを入手して、ローカル (またはオンプレミス) に保存できます。
これが該当し、AWSSDK アセンブリをまだ入手していない場合は、「AWSSDK アセンブリを入手する」を参照してください。ローカルに保存されたアセンブリの使用方法については、「NuGet を使用せずに AWSSDK アセンブリをインストールする」を参照してください。

コマンドプロンプトまたはターミナルからの NuGet の使用

1. NuGet の AWSSDK パッケージに移動し、プロジェクトに必要なパッケージ (AWSSDK.S3など) を決定します。
2. そのパッケージのウェブページから.NET CLI コマンドをコピーします。以下に例を示します。

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

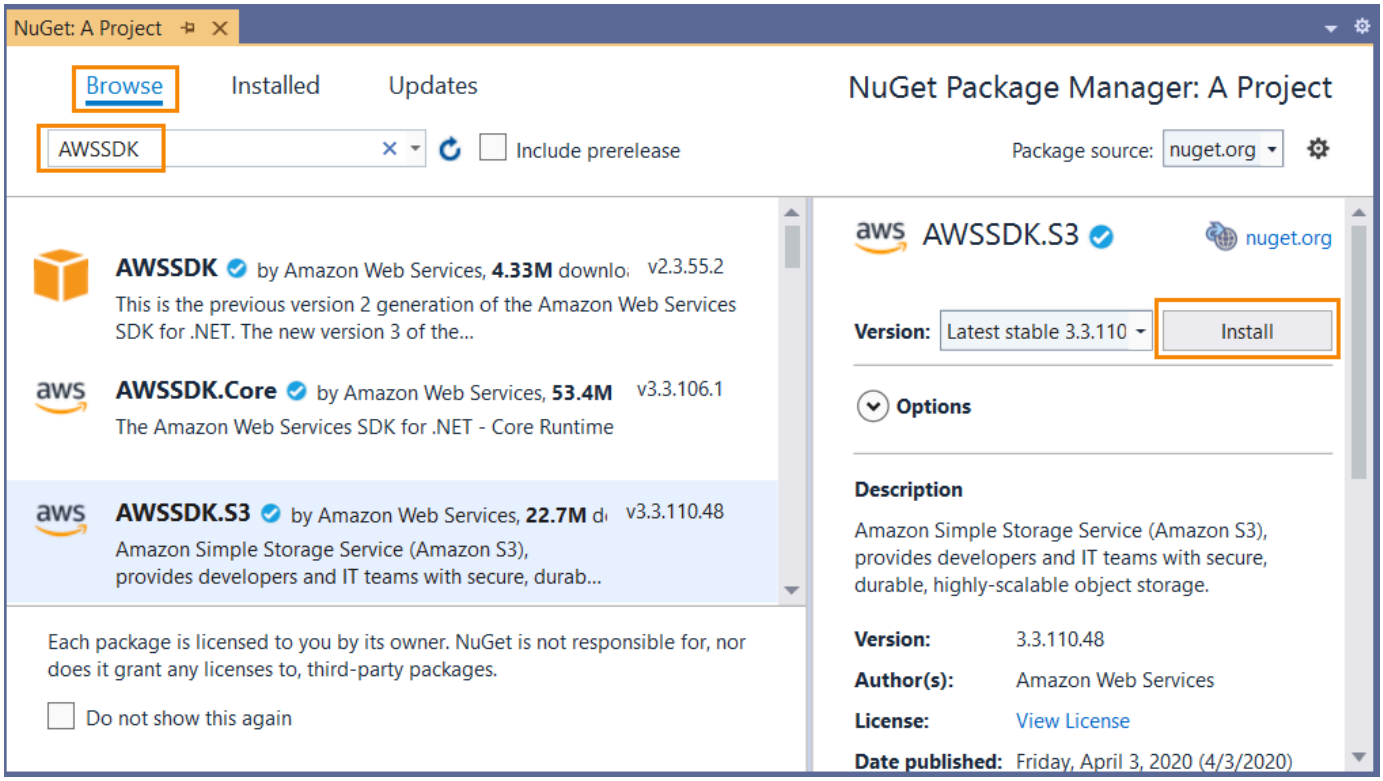
3. プロジェクトのディレクトリで、その .NET CLI コマンドを実行します。NuGet は、AWSSDK.Core などの依存関係があるものもすべてインストールします。

> **Note**
>
> 最新バージョンの NuGet パッケージのみが必要な場合は、以下の例に示すように、コマンドからバージョン情報を除外できます。
>
> ```
> dotnet add package AWSSDK.S3
> ```

Visual Studio ソリューションエクスプローラーからの NuGet の使用

1. ソリューションエクスプローラーで、プロジェクトを右クリックして、コンテキストメニューの [NuGet パッケージの管理] を選択します。
2. NuGet パッケージマネージャーの左側のペインで、[参照] を選択します。検索ボックスを使用して、インストールするパッケージを検索できます。NuGet は、AWSSDK.Core などの依存関係があるものもすべてインストールします。

   次の図は、AWSSDK.S3 パッケージのインストールを示しています。



パッケージマネージャーコンソールからの NuGet の使用

Visual Studio で [Tools] (ツール) メニューから、[Package Manager Console] (NuGet パッケージマネージャー)、[Package Manager Console] (パッケージマネージャーコンソール) を選択します。

パッケージマネージャーコンソールで **`Install-Package`** コマンドを使用して、必要な AWSSDK パッケージをインストールできます。たとえば、AWSSDK.S3 をインストールするには、次のコマンドを使用します。

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

NuGet は、AWSSDK.Core などの依存関係があるものもすべてインストールします。

以前のバージョンのパッケージをインストールする必要がある場合は、`-Version` オプションを使用して希望するパッケージのバージョンを指定します。以下に例を示します。

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

Package Manager コンソールのコマンドの詳細については、Microsoft のNuGet のドキュメントの「PowerShell リファレンス」を参照してください。

NuGet を使用せずに AWSSDK アセンブリをインストールする

このトピックでは、「AWSSDK アセンブリを入手する」の説明に従って入手し、ローカル (またはオンプレミス) に保存した AWSSDK アセンブリの使用方法について説明します。この方法は、SDK リファレンスの処理には推奨されませんが、一部の環境では必須です。

> **Note**
>
> SDK リファレンスを処理するには、各プロジェクトで必要とされる NuGet パッケージのみをダウンロードしてインストールする方法が推奨されます。その方法については、「NuGet を使用して AWSSDK パッケージをインストールする」を参照してください。

AWSSDK アセンブリをインストールするには

1. 必要な AWSSDK アセンブリのプロジェクト領域にフォルダを作成します。この例では、フォルダに `AwsAssemblies` と名前を付けます。
2. AWSSDK アセンブリをまだ入手していない場合は、AWSSDK アセンブリを入手します。アセンブリは、ローカルのダウンロードフォルダまたはインストールフォルダに保存されます。必要なアセンブリの DLL ファイルを、そのダウンロードフォルダからプロジェクト (この例の場合、`AwsAssemblies` フォルダ) にコピーします。

   必ず依存関係もコピーします。依存関係については、GitHub ウェブサイトを参照してください。
3. 次のように、必要なアセンブリを参照します。

   Cross-platform development

   1. プロジェクトの `.csproj` ファイルを開き、`<ItemGroup>` 要素を追加します。
   2. `<ItemGroup>` 要素では、必要なアセンブリごとに `Include` 属性を持つ `<Reference>` 要素を追加します。

      たとえば、Amazon S3 の場合は、次の行をプロジェクトの `.csproj` ファイルに追加します。

      Linux および macOS の場合:

      ```
      <ItemGroup>
      ```

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

Windows の場合:

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

3. プロジェクトの `.csproj` ファイルを保存します。

Windows with Visual Studio and .NET Core

1. Visual Studio で、プロジェクトをロードして、プロジェクトを開き、リファレンスを追加します。
2. ダイアログボックスの下部にある [Browse] (参照) ボタンをクリックします。プロジェクトのフォルダと必要な DLL ファイルをコピーしたサブフォルダ (`AwsAssemblies` など) に移動します。
3. すべての DLL ファイルを選択し、[Add] (追加)、[OK] の順に選択します。
4. プロジェクトを保存します。

の AWS リージョンの設定 AWS SDK for .NET

AWS リージョンを使用すると、特定の地理的リージョンに物理的に存在する AWS サービスにアクセスできます。これは、冗長性と、ユーザーがアクセスする場所の近くでのデータとアプリケーションの実行を維持するために有効です。

各 AWS サービスでサポートされているすべてのリージョンとエンドポイントの現在のリストを表示するには、の「サービスエンドポイントとクォータ」を参照してくださいAWS 全般のリファレンス。既存のリージョンエンドポイントのリストを表示するには、AWS サービスエンドポイントを参照してください。リージョンの詳細については、「アカウントで使用できる AWS リージョンを指定する」を参照してください。

特定のリージョンに移動する AWS サービスクライアントを作成できます。すべての AWS サービスクライアントに使用されるリージョンでアプリケーションを設定することもできます。次に、この 2 つのケースについて説明します。

特定のリージョンを使用してサービスクライアントを作成する

アプリケーション内の任意の AWS サービスクライアントのリージョンを指定できます。この方法でのリージョンの設定は、特定のサービスクライアントのグローバル設定よりも優先されます。

既存のリージョン

この例では、既存のリージョンで Amazon EC2 クライアントをインスタンス化する方法を示します。ここでは、定義済み RegionEndpoint フィールドを使用します。

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

RegionEndpoint クラスを使用した新しいリージョン

この例では、RegionEndpoint.GetBySystemName を使用して新しいリージョンエンドポイントを作成する方法を示しています。

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

サービスクライアント設定クラスを使用した新しいリージョン

この例では、サービスクライアント設定クラスの ServiceURL プロパティを使用してリージョンを指定する方法を示しています (この例では、Amazonec2Config クラスを使用)。

この手法は、エンドポイントリージョンが通常のエンドポイントパターンに従っていない場合でも有効です。

```
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
}
```

すべてのサービスクライアント用にリージョンを指定する

アプリケーションが作成するすべての AWS サービスクライアントにリージョンを指定する方法はいくつかあります。このリージョンは、特定のリージョンを使用して作成されていないサービスクライアントに使用されます。

は、次の順序でリージョン値 AWS SDK for .NET を検索します。

プロファイル

アプリケーションまたは SDK によってロードされたプロファイルを設定します。詳細については、「認証情報とプロファイルの解決」を参照してください。

環境変数

以下のように `AWS_REGION` 環境変数を設定します。

Linux または macOS の場合:

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

Windows の場合:

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

> **Note**
>
> この環境変数をシステム全体に設定すると (`export` または `setx` を使用)、 AWS SDK for .NETだけではなく、すべての SDK とツールキットが影響を受けます。

AWSConfigs クラス

以下のように AWSConfigs.AWSRegion プロパティとして設定します。

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

リージョン解像度

上記のいずれの方法も の指定に使用されない場合 AWS リージョン、 は AWS サービスクライアントが動作するリージョンの検索 AWS SDK for .NET を試みます。

リージョン解像度オーダー

1. app.configやweb.configなどのアプリケーション設定ファイル。
2. 環境変数 (AWS_REGIONおよびAWS_DEFAULT_REGION)
3. AWSConfigs.AWSProfileName の値で指定された名前のプロファイル。
4. AWS_PROFILE 環境変数で指定された名前を持つプロファイル。
5. [default] のプロファイル。
6. Amazon EC2 インスタンスのメタデータ (EC2 インスタンスで実行されている場合)。

リージョンが見つからない場合、SDK は AWS サービスクライアントにリージョンが設定されていないことを示す例外をスローします。

中国 (北京) リージョンに関する特別な情報

中国 (北京) リージョンでサービスを使用するには、中国 (北京) リージョン固有のアカウントと認証情報が必要です。他の AWS リージョンのアカウントと認証情報は、中国 (北京) リージョンでは機能しません。同様に、中国 (北京) リージョンのアカウントと認証情報は、他の AWS リージョンでは機能しません。中国 (北京) リージョンで利用可能なエンドポイントとプロトコルの詳細については、「北京リージョンのエンドポイント」を参照してください。

新しい AWS サービスに関する特別な情報

新しい AWS サービスは、最初にいくつかのリージョンで起動し、その後他のリージョンでサポートされます。このような場合、そのサービス用の新しいリージョンにアクセスするために最新の SDK をインストールする必要はありません。新しく追加されたリージョンは、前述したように、クライアントごと、またはグローバルに指定できます。

認証情報とプロファイルの解決

は特定の順序で認証情報 AWS SDK for .NET を検索し、現在のアプリケーションで使用可能な最初のセットを使用します。

認証情報の検索順序

1. 「」で説明されているように、 AWS サービスクライアントで明示的に設定される認証情報アプリケーションでの認証情報とプロファイルへのアクセス。

   > **Note**
   >
   > このトピックは推奨される認証情報の指定方法ではないため、特別な考慮事項 セクションに記載されています。

2. 3 つの変数すべてに値がある場合AWS_ACCESS_KEY_ID、、AWS_SECRET_ACCESS_KEY、およびAWS_SESSION_TOKEN環境変数から作成される SessionAWSCredentials。
3. 両方の変数に値がある場合、 AWS_ACCESS_KEY_IDおよび AWS_SECRET_ACCESS_KEY環境変数から作成される BasicAWSCredentials。
4. 両方の変数に値がある場合、 AWS_WEB_IDENTITY_TOKEN_FILEおよび AWS_ROLE_ARN環境変数から作成される AssumeRoleWithWebIdentityCredentials。
5. AWSConfigs.AWSProfileName の値で指定された名前を持つ認証情報プロファイル。
6. AWS_PROFILE 環境変数で指定された名前を持つ認証情報プロファイル。
7. [default] 認証情報プロファイル。
8. コンテナ認証情報プロバイダー。
9. Amazon EC2 インスタンスメタデータ。

アプリケーションが本番環境などの Amazon EC2 インスタンスで実行されている場合は、「IAM ロールを使用したアクセス権の付与」の説明に従って IAM ロールを使用します。プレリリーステストなど、それ以外の場合は、ウェブアプリケーションがサーバー上でアクセスできる認証情報ファイル形式を使用するファイルに AWS 認証情報を保存します。

環境変数と認証情報プロファイルの詳細については、AWS SDKs」の次のトピックを参照してください。環境変数、環境変数リスト、共有設定ファイルと認証情報ファイル。

プロファイルの解決

認証情報の 2 つの異なるストレージメカニズムでは、それらを使用する AWS SDK for .NET ようにを設定する方法を理解することが重要です。AWSConfigs.AWSProfilesLocation プロパティは、 が認証情報プロファイル AWS SDK for .NET を検索する方法を制御します。

AWSProfilesLocation	プロファイルの解決動作
null (未設定) または空	プラットフォームでサポートされていれば SDK ストアを検索し、次にデフォルトの場所にある共有 AWS 認証情報ファイルを検索します。プロファイルがいずれの場所にもない場合は、`~/.aws/config` (Linux または macOS) または `%USERPROFILE%\.aws\config` (Windows) を検索します。
AWS 認証情報ファイル形式のファイルへのパス	指定されたファイルのみを対象に、指定された名前のプロファイルを検索します。

フェデレーティッドユーザーアカウントの認証情報の使用

AWS SDK for .NET （ AWSSDK.Core バージョン 3.1.6.0 以降) を使用するアプリケーションは、Active Directory フェデレーションサービス (AD FS) を介してフェデレーティッドユーザーアカウントを使用して、Security Assertion Markup Language (SAML) を使用して AWS サービスにアクセスできます。

フェデレーティッドアクセスサポートでは、ユーザーは Active Directory を使用して認証できます。一時的な認証情報は、自動的にユーザーに許可されます。これらの一時的な認証情報は 1 時間有効で、アプリケーションが AWS サービスを呼び出すときに使用されます。一時的な認証情報の管理は SDK によって処理します。ドメイン結合されたユーザーアカウントでは、アプリケーションが呼び出しを行ったときに資格情報の有効期限が切れている場合に、そのユーザーは自動的に再認証され、新しい認証情報が付与されます （ ドメインに参加していないアカウントの場合、ユーザーは再認証の前に認証情報を入力するように求められます ）。

このサポートを .NET アプリケーションで使用するには、まず PowerShell コマンドレットを使用してロールプロファイルをセットアップする必要があります。方法については、AWS Tools for Windows PowerShell のドキュメントを参照してください。

ロールプロファイルを設定したら、アプリケーションでプロファイルを参照します。そのための方法はいくつかありますが、その 1 つとして AWSConfigs.AWSProfileName プロパティを他の認証情報プロファイルと同じように設定する方法があります。

AWS Security Token Service アセンブリ (AWSSDK.SecurityToken) は、 AWS 認証情報を取得するための SAML サポートを提供します。フェデレーティッドユーザーアカウントの認証情報を使用するには、アプリケーションでこのアセンブリが使用できることを確認してください。

ロールまたは一時認証情報の指定

Amazon EC2 インスタンスで実行されるアプリケーションの場合、認証情報を管理する最も安全な方法は、「IAM ロールを使用したアクセス権の付与」での説明に従って IAM ロールを使用することです。

組織外部のユーザーに対してソフトウェア実行可能ファイルが利用可能になるアプリケーションシナリオでは、一時的なセキュリティ認証情報を使用するようにソフトウェアを設計することをお勧めします。 AWS リソースへのアクセスを制限するだけでなく、これらの認証情報は指定された期間後に期限切れになるという利点があります。一時的なセキュリティ認証情報の使用方法の詳細については、以下を参照してください。

- 一時的な認証情報
- Amazon Cognito アイデンティティプール

プロキシ認証情報の使用

ソフトウェアがプロキシ AWS を介して と通信する場合は、サービスの Configクラスの ProxyCredentialsプロパティを使用して、プロキシの認証情報を指定できます。サービスの Config クラスは通常、サービスのプライマリ名前空間の一部です。例として、Amazon.CloudDirectory 名前空間の AmazonCloudDirectoryConfig や Amazon.GameLift 名前空間の AmazonGameLiftConfig などがあります。

たとえば、Amazon S3 の場合は、次のようなコードを使用できます。SecurelyStoredUserName と SecurelyStoredPassword は、NetworkCredential オブジェクトで指定されたプロキシ ユーザー名とパスワードです。

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

> **Note**
>
> SDK の以前のバージョンでは `ProxyUsername` および `ProxyPassword` が使用されていましたが、これらのプロパティは非推奨になりました。

再試行とタイムアウト

AWS SDK for .NET では、 AWS サービスへの HTTP リクエストの再試行回数とタイムアウト値を設定できます。再試行とタイムアウトのデフォルト値がアプリケーションで適切でない場合は、特定の要件に対してそれらの値を調整できますが、それによってアプリケーションの動作にどのように影響するかを理解しておくことが重要です。

再試行とタイムアウトに使用する値を決定するには、以下の点を検討します。

- ネットワーク接続が低下したり、 AWS サービスにアクセスできない場合、 AWS SDK for .NET とアプリケーションはどのように応答する必要がありますか? 呼び出しがすぐに失敗するか、またはユーザーに代わって呼び出しが再試行され続けるか、どちらが適切か。
- 応答性が必要なユーザー対応アプリケーションまたはウェブサイトであるか、またはレイテンシーの増加に耐性があるバックグラウンド処理ジョブであるか。
- アプリケーションが低レイテンシーの信頼性の高いネットワークでデプロイされているか、または信頼性が低い接続でリモートの場所にデプロイされているか。

再試行

概要:

AWS SDK for .NET は、サーバー側のスロットリングまたは接続の切断が原因で失敗したリクエストを再試行できます。サービス設定クラスの 2 つのプロパティを使用して、サービスクライアントの再試行動作を指定します。サービス設定クラスは、AWS SDK for .NET API リファレンスの抽象 Amazon.Runtime.ClientConfig クラスからこれらのプロパティを継承します。

- `RetryMode` では、Amazon.Runtime.RequestRetryMode 列挙で定義される 3 つの再試行モードのいずれかを指定します。

  アプリケーションのデフォルト値は、`AWS_RETRY_MODE`環境変数または共有設定ファイルの retry_mode AWS 設定を使用して制御できます。

- `MaxErrorRetry` では、サービスクライアントレベルで許可される再試行の回数を指定します。SDK では、指定した回数だけ操作が再試行された後に、失敗となり、例外がスローされます。

  アプリケーションのデフォルト値は、 `AWS_MAX_ATTEMPTS`環境変数または共有設定ファイルの max_attempts AWS 設定を使用して制御できます。

これらのプロパティの詳細な説明は、AWS SDK for .NET API リファレンスの抽象 Amazon.Runtime.ClientConfig クラスに記載されています。`RetryMode` の各値は、次の表に示すように、デフォルトでは `MaxErrorRetry` の特定の値に対応します。

RetryMode	Corresponding MaxErrorRetry (Amazon DynamoDB)	Corresponding MaxErrorRetry (all others)
Legacy	10	4
Standard	10	2
Adaptive (experimental)	10	2

行動

アプリケーションの起動時

アプリケーションの起動時に、`RetryMode` および `MaxErrorRetry` のデフォルト値が SDK によって設定されます。これらのデフォルト値は、他の値を指定しない限り、サービスクライアントを作成するときに使用されます。

- 環境でプロパティが設定されていない場合は、`RetryMode` のデフォルトが Legacy として設定され、`MaxErrorRetry` のデフォルトが上記の表の対応する値で設定されます。
- 環境で再試行モードが設定されている場合は、その値が `RetryMode` のデフォルトとして使用されます。`MaxErrorRetry` のデフォルトは、最大エラー数の値も環境で設定されていない限り、上記の表の対応する値で設定されます (次の説明を参照)。
- 環境で最大エラー数の値が設定されている場合は、その値が `MaxErrorRetry` のデフォルトとして使用されます。Amazon DynamoDB はこのルールの例外です。`MaxErrorRetry` のデフォルトの DynamoDB 値は、常に上記の表の値になります。

アプリケーションの実行時

サービスクライアントを作成すると、RetryMode および MaxErrorRetry の次のデフォルト値を使用できます。または、前述したように、他の値を指定することもできます。他の値を指定するには、サービスクライアントを作成する際に、AmazonDynamoDBConfig や AmazonSQSConfig など、サービス設定オブジェクトを作成して含めます。

サービスクライアントの作成後にこれらの値を変更することはできません。

考慮事項

再試行が発生すると、リクエストのレイテンシーが増加します。アプリケーションでのリクエストレイテンシーの合計とエラー発生率の制限に基づいて、再試行回数を設定する必要があります。

タイムアウト

AWS SDK for .NET では、サービスクライアントレベルおよびメソッド呼び出しごとにリクエストタイムアウトを設定できます。タイムアウトを設定するには 2 つのメカニズムがあり、以降のセクションで説明します。

- 非同期呼び出しを使用している場合は、メソッドの CancellationToken パラメータを使用できます。
- .NET Framework で同期呼び出しを使用している場合は、抽象 Amazon.Runtime.ClientConfig クラスの Timeout および ReadWriteTimeout プロパティを使用できます。

タイムアウトに **CancellationToken** パラメータを使用する

AWS SDK for .NET を使用すると、 CancellationTokenパラメータを使用して非同期呼び出しのリクエストタイムアウトを設定できます。次のコードスニペットは例を示しています。リクエストが 10 秒以内に完了しなかった場合、コードは System.Threading.Tasks.TaskCanceledException をスローします。

```
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);
}
```

タイムアウトに **Timeout** および **ReadWriteTimeout** プロパティを使用する

> **Note**
>
> Timeout プロパティは非同期呼び出しには影響しません。非同期呼び出しを使用している場合は、代わりに「タイムアウトに CancellationToken パラメータを使用する」を参照してください。

AWS SDK for .NET では、リクエストタイムアウトとソケットの読み取り/書き込みタイムアウト値をサービスクライアントレベルで設定できます。これらの値は、抽象 Amazon.Runtime.ClientConfig クラスの Timeout プロパティと ReadWriteTimeout プロパティで指定されます。これらの値は、 AWS サービスクライアントオブジェクトによって作成された HttpWebRequest オブジェクトの Timeoutおよび ReadWriteTimeoutプロパティとして渡されます。デフォルトでは、Timeout の値は 100 秒であり、ReadWriteTimeout の値は 300 秒です。

ネットワークのレイテンシーが大きい場合、または操作が再試行される条件が存在する場合に、長いタイムアウト値と大きい再試行回数を使用すると、一部の SDK 操作が応答していないように見えることがあります。

> **Note**
>
> ポータブルクラスライブラリ (PCL) をターゲット AWS SDK for .NET とする のバージョンは、 クラスではなく HttpClient HttpWebRequest クラスを使用し、タイムアウトプロパティのみをサポートします。

デフォルトのタイムアウト値の例外を次に示します。これらの値は、タイムアウト値を明示的に設定するとオーバーライドされます。

- AmazonS3Client.PutObjectAsync()、AmazonS3Client.UploadPartAsync()、AmazonGlacierClient.UploadA など、呼び出されるメソッドによってストリームがアップロードされると、Timeout および ReadWriteTimeout は最大値に設定されます。
- .NET Framework をターゲット AWS SDK for .NET とする のバージョンはTimeout、すべての AmazonS3Client および AmazonGlacierClient オブジェクトの最大値ReadWriteTimeoutに と を設定します。
- ポータブルクラスライブラリ (PCL) と .NET Core をターゲット AWS SDK for .NET とする のバージョンは、すべての AmazonS3Client および AmazonGlacierClient オブジェクトの最大値Timeoutに設定されます。

次の例では、標準再試行モード、3 回の最大再試行、10 秒のタイムアウト、および 10 秒の読み取り/書き込みタイムアウト (該当する場合) を指定する方法を示しています。AmazonS3Client コンストラクタには AmazonS3Config オブジェクトが指定されています。

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

オブザーバビリティ

オブザーバビリティとは、システムが出力するデータから、そのシステムの現在の状態をどの程度推測できるかという指標のことです。出力されるデータは、一般的にテレメトリと呼ばれます。

は、2 つの一般的なテレメトリシグナル、メトリクスとトレース、およびログ記録を提供 AWS SDK for .NET できます。TelemetryProvider をワイヤアップして、テレメトリデータをオブザーバビリティバックエンド (AWS X-Ray や Amazon CloudWatch など) に送信し、それに対応できます。

デフォルトでは、テレメトリ信号は SDK で無効になっています。このトピックでは、テレメトリ出力を有効にして設定する方法について説明します。

その他のリソース

オブザーバビリティの有効化と使用の詳細については、以下のリソースを参照してください。

- OpenTelemetry
- ブログ記事OpenTelemetry AWS SDK for .NET を使用した のオブザーバビリティの強化」。
- このブログ記事では、 AWS .NET OpenTelemetry ライブラリの一般提供を発表しています。
- OpenTelemetry のエクスポーター
- のオブザーバビリティの例については AWS Tools for PowerShell、「Tools for PowerShell ユーザーガイド」の「オブザーバビリティ」を参照してください。

`TelemetryProvider` の設定

次の例に示すように、すべてのサービスクライアントまたは個々のクライアントについて、アプリケーションで `TelemetryProvider` をグローバルに設定できます。the section called “テレメトリプロバイダー” セクションには、SDK で提供される実装に関する情報など、テレメトリの実装に関する情報が含まれています。

デフォルトのグローバルテレメトリプロバイダーを設定する

デフォルトでは、すべてのサービスクライアントはグローバルに利用可能なテレメトリプロバイダーの使用を試みます。これにより、プロバイダーを 1 回設定でき、すべてのクライアントがそれを使用します。これは、サービスクライアントを作成する前に 1 回だけ行う必要があります。

次のコードスニペットは、グローバルテレメトリプロバイダーを設定する方法を示しています。次に、Amazon S3 サービスクライアントを作成し、オペレーションの実行を試みますが、失敗します。このコードは、トレースとメトリクスの両方をアプリケーションに追加します。このコードは、`OpenTelemetry.Exporter.Console` および `OpenTelemetry.Instrumentation.AWS` の NuGet パッケージを使用します。

> **Note**
>
> 認証 AWS IAM アイデンティティセンター に を使用している場合は、必ず `AWSSDK.SSO`とも追加してください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();
```

特定のサービスクライアントのテレメトリプロバイダーを設定する

個々のサービスクライアントは、特定のテレメトリプロバイダー (グローバルプロバイダーを除く) で設定できます。これを行うには、サービスクライアントコンストラクタの Config オブジェクトの TelemetryProvider クラスを使用します。例えば、AmazonS3Config を参照して、TelemetryProvider プロパティを探します。カスタムテレメトリの実装については、「the section called “テレメトリプロバイダー”」を参照してください。

トピック

- メトリクス
- テレメトリプロバイダー

メトリクス

次の表に、SDK が出力するテレメトリメトリクスを示します。メトリクスを観測可能にするようにテレメトリプロバイダーを設定します。

どのようなメトリクスが出力されますか？

メトリクス名	単位	タイプ	属性	説明
client.call.duration	s	ヒストグラム	rpc.service、rpc.method	全体的な通話時間 (再試行と、リクエストとレスポンス本文の送受信時間を含む)
client.uptime	s	ヒストグラム	rpc.service	クライアントが作成されてからの時間
client.call.attempts	{attempt}	MonotonCounter	rpc.service、rpc.method	個々のオペレーションの試行回数
client.call.errors	{error}	MonotonCounter	rpc.service、rpc.method、 exception.type	オペレーションのエラーの数
client.call.attempt_duration	s	ヒストグラム	rpc.service、rpc.method	サービスへの接続、リクエストの送信、HTTP ステータスコードとヘッダーの取得にかかる時間 (キューに入れられた送信待機時間を含む)
client.call.resolve_endpoint_duration	s	ヒストグラム	rpc.service、rpc.method	リクエストのエンドポイント (DNS ではなくエンドポイントリゾルバー) の解決にかかる時間
client.call.serialization_duration	s	ヒストグラム	rpc.service、rpc.method	メッセージ本文のシリアル化にかかる時間
client.call.deserialization_duration	s	ヒストグラム	rpc.service、rpc.method	メッセージ本文の逆シリアル化にかかる時間

メトリクス名	単位	タイプ	属性	説明
client.call.auth.signing_duration	s	ヒストグラム	rpc.service、rpc.method	リクエストの署名にかかる時間
client.call.auth.resolve_identity_duration	s	ヒストグラム	rpc.service、rpc.method	ID プロバイダーから ID (AWS 認証情報やベアラートークンなど) を取得するのにかかる時間
client.http.bytes_sent	方法	Monotonic Counter	server.address	HTTP クライアントによって送信された合計バイト数
client.http.bytes_received	方法	Monotonic Counter	server.address	HTTP クライアントが受信した合計バイト数

列の説明は次のとおりです。

- メトリクス名 – 出力されたメトリクスの名前。
- Units – メトリクスの測定単位。単位は、UCUM の大文字と小文字を区別する (「c/s」) 表記で指定します。
- Type – メトリクスのキャプチャに使用される計測器のタイプ。
- 属性 – メトリクスで出力される属性 (ディメンション) のセット。
- 説明 - メトリクスが測定する内容の説明。

テレメトリプロバイダー

SDK は、OpenTelemetry の実装をテレメトリプロバイダーとして提供します。これについては、次のセクションで説明します。

特定のテレメトリ要件がある場合、テレメトリソリューションがすでに考慮されている場合、またはテレメトリデータのキャプチャと処理方法をきめ細かく制御する必要がある場合は、独自のテレメトリプロバイダーを実装することもできます。

TelemetryProvider クラスに独自の実装を登録します。以下は、独自の `TracerProvider`と を登録する方法の簡単な例です`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());
```

トピック

- OpenTelemetry ベースのテレメトリプロバイダーを設定する

OpenTelemetry ベースのテレメトリプロバイダーを設定する

には、OpenTelemetry ベースのテレメトリプロバイダーの実装 AWS SDK for .NET が含まれています。このプロバイダーをグローバルテレメトリプロバイダーとして設定する方法の詳細については、「TelemetryProvider の設定」を参照してください。このテレメトリプロバイダーを使用するには、プロジェクトに次のリソースが必要です。

- OpenTelemetry.Instrumentation.AWS NuGet パッケージ。
- OTLP やコンソールなどのテレメトリエクスポーター。詳細については、OpenTelemetry ドキュメントの「エクスポーター」を参照してください。

SDK に含まれる OpenTelemetry 実装は、HTTPS リクエスト、認証情報、および圧縮のトレース量を減らすように設定できます。これを行うには、次のように、SuppressDownstreamInstrumentation オプションを true に設定します。

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

```
    .Build();
```

このプロバイダーの詳細については、OpenTelemetry AWS SDK for .NET を使用した のブログ記事「Enhancing Observability」を参照してください。

ユーザーとロールに関する追加情報

で .NET 開発を実行する場合、 AWS または で .NET アプリケーションを実行する場合は AWS、これらのタスクに適したユーザー、アクセス許可セット、サービスロールの組み合わせが必要です。

作成する特定のユーザー、権限セット、サービス ロール、それらの使用方法は、アプリケーションの要件によって異なります。それらが使用される理由とその作成方法について、以下でさらに説明します。

ユーザーとアクセス許可セット

長期的な認証情報を持つ IAM ユーザーアカウントを使用して AWS のサービスにアクセスすることは可能ですが、これはもはやベストプラクティスではなく、避ける必要があります。開発中であっても、 でユーザーとアクセス許可セットを作成し AWS IAM アイデンティティセンター 、ID ソースによって提供される一時的な認証情報を使用することがベストプラクティスです。

開発には、自分で作成したユーザー、または による認証 AWS で付与されたユーザーを使用できます。適切な AWS マネジメントコンソール アクセス許可がある場合は、そのユーザーに対して最小特権で異なるアクセス許可セットを作成したり、開発プロジェクト専用の新しいユーザーを作成したりして、最小特権でアクセス許可セットを提供したりできます。選択する行動方針 (ある場合) は、状況によって異なります。

これらのユーザーとアクセス許可セット、および作成方法の詳細については、「AWS SDK とツール リファレンスガイド」の「認証とアクセス」と、「AWS IAM アイデンティティセンター ユーザーガイド」の「はじめに」を参照してください。

サービス役割

ユーザーに代わって AWS サービスにアクセスするための AWS サービスロールを設定できます。このタイプのアクセスは、複数のユーザーがアプリケーションをリモートで実行する場合 (たとえば、この目的のために作成した Amazon EC2 インスタンス上で実行する場合など) に適しています。

サービスロールを作成するプロセスは状況によって異なりますが、基本的に次のプロセスを実行します。

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/iam/:// www.com」で IAM コンソールを開きます。
2. [ロール]、[ロールの作成] の順に選択します。
3. [AWS service] (AWS サービス) を選択して、[EC2] (この例の場合) を見つけて選択し、[EC2] ユースケース (この例の場合) を選択します。
4. 次へ: アクセス許可を選択し、アプリケーションが使用する AWS サービスに適したポリシーを選択します。

   > **Warning**
   >
   > AdministratorAccess ポリシーは選択しないでください。このポリシーを選択すると、お使いのアカウント内のほぼすべての情報の読み取りと書き込みのアクセス許可が有効になります。

5. [Next: Tags] (次へ: タグ) を選択して、必要なタグをすべて入力します。

   タグに関する情報は、IAM ユーザーガイドのAWS リソースタグを使用してアクセスを制御するで確認できます。
6. [Next: Review] (次の手順: 確認) を選択し、ロール名とロールの説明を入力します。次に、[Create role (ロールの作成)] を選択します。

IAM ロールに関する概要については、IAM ユーザーガイドの「ID ((ユーザー、グループ、ロール)」を参照してください。ロールの詳細については、同ガイドの「IAM ロール」トピックを参照してください。

ロールに関する追加情報

- Amazon Elastic Container Service (Amazon ECS) タスクで、タスク用の IAM ロールを使用します。
- Amazon EC2 インスタンスで実行中のアプリケーションに対して、IAM ロールを使用します。

AWS SDK for .NET プロジェクトの高度な設定

このセクションのトピックでは、お客様が関心を持つ可能性のある追加の設定タスクやメソッドについて説明します。

トピック

- AWSSDK.Extensions.NETCore.Setup および IConfiguration インターフェイスの使用
- 他のアプリケーションパラメータの設定
- の設定ファイルリファレンス AWS SDK for .NET

AWSSDK.Extensions.NETCore.Setup および IConfiguration インターフェイスの使用

(このトピックは、以前は「.NET Core AWS SDK for .NET で を設定する」というタイトルでした)

.NET Core における最も大きな変更の 1 つは、`ConfigurationManager` と標準の `app.config` および `web.config` ファイルが削除されたことです。これらは .NET Framework および ASP.NET アプリケーションで使用されていました。

.NET Core での設定は、設定プロバイダーによって定められたキーと値のペアに基づいて行われます。設定プロバイダーは、コマンドライン引数、ディレクトリファイル、環境変数、設定ファイルなど、さまざまな設定ソースから設定データをキーと値のペアに読み込みます。

> **Note**
>
> 詳細については、「ASP.NET Core の構成」を参照してください。

.NET Core AWS SDK for .NET で を簡単に使用するために、AWSSDK.Extensions.NETCore.Setup NuGet パッケージを使用できます。多くの .NET Core ライブラリと同様に、インターフェイスに拡張機能メソッドを追加して、 AWS 設定をシームレスに`IConfiguration`します。

このパッケージのソースコードは GitHub の https://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.NETCore.Setup にあります。

AWSSDK.Extensions.NETCore.Setup の使用

ASP.NET コアモデル-ビューコントローラー (MVC) アプリケーションを作成するとします。このアプリケーションは、Visual Studio で ASP.NET Core ウェブアプリケーションテンプレートを使用するか、または .NET Core CLI で `dotnet new mvc ...` を実行することにより作成できます。アプリケーションを作成すると、`Startup.cs` のコンストラクタは `appsettings.json` などの設定プロバイダーからさまざまな入力ソースを読み取ることによって設定を処理します。

```
public Startup(IConfiguration configuration)
```

```
{
    Configuration = configuration;
}
```

Configuration オブジェクトを使用して AWS のオプションを取得するには、まず AWSSDK.Extensions.NETCore.Setup NuGet パッケージを追加します。次に、以下の記載に従って設定ファイルにオプションを追加します。

プロジェクトに追加されたファイルの 1 つに、appsettings.Development.json があります。このファイルは、EnvironmentName が Development に設定された場合に対応します。開発時にこのファイルに設定を入力すると、ローカルテスト中にのみ読み取られます。を Production にEnvironmentName設定した Amazon EC2 インスタンスをデプロイすると、このファイルは無視され、 は Amazon EC2 インスタンス用に設定された IAM 認証情報とリージョンに AWS SDK for .NET フォールバックされます。

次の設定は、appsettings.Development.json 設定を指定するためにプロジェクトの AWS ファイルに追加できる値の例を示しています。

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

CSHTML ファイルの設定にアクセスするには、Configuration ディレクティブを使用します。

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

<h1>Contact</h1>

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

コードから ファイルで設定された AWS オプションにアクセスするには、 に追加されたGetAWSOptions拡張メソッドを呼び出しますIConfiguration。

これらのオプションからサービスクライアントを構築するには、CreateServiceClient を呼び出します。次の例は、Amazon S3 サービスクライアントを作成する方法を示しています。(必ず AWSSDK.S3 NuGet パッケージをプロジェクトに追加してください)

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

appsettings.Development.json ファイルで複数のエントリを使用して、互換性のない設定を持つ複数のサービスクライアントを作成することもできます。次の例に示すように、service1 の設定には us-west-2 リージョンが含まれ、service2 の設定には特殊なエンドポイント URL が含まれています。

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

その後、JSON ファイルのエントリを使用することで、特定のサービスのオプションを取得できます。例えば、service1 の設定を取得するには以下を使用します。

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

appsettings ファイルで指定できる値

appsettings.Development.json ファイルには次のアプリケーション設定値を設定できます。フィールド名は、記載された大文字小文字を区別して使用する必要があります。これらの設定の詳細については、「AWS.Runtime.ClientConfig」クラスを参照してください。

- Region
- Profile
- ProfilesLocation
- SignatureVersion
- RegionEndpoint

- UseHttp
- ServiceURL
- AuthenticationRegion
- AuthenticationServiceName
- MaxErrorRetry
- LogResponse
- BufferSize
- ProgressUpdateInterval
- ResignRetries
- AllowAutoRedirect
- LogMetrics
- DisableLogging
- UseDualstackEndpoint

ASP.NET Core の依存関係インジェクション

AWSSDK.Extensions.NETCore.Setup NuGet パッケージには、ASP.NET Core の新しい依存関係インジェクションシステムも組み込まれています。アプリケーションの `Startup` クラスの `ConfigureServices` メソッドは、MVC サービスが追加される場所です。アプリケーションが Entity Framework を使用している場合は、これが初期化される場所でもあります。

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

> **Note**
>
> .NET Core の依存関係インジェクションの背景については、.NET Core のドキュメントサイトを参照してください。

`AWSSDK.Extensions.NETCore.Setup` NuGet パッケージは、依存関係インジェクションに AWS サービスを追加するために`IServiceCollection`使用できる新しい拡張機能メソッドを に追加

します。次のコードは、読み取り元の AWS オプションを追加してAmazon S3と DynamoDB をサービスのリストIConfigurationに追加する方法を示しています。(必ず AWSSDK.S3 および AWSSDK.DynamoDBv2 NuGet パッケージをプロジェクトに追加してください)

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

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

ここで、MVC コントローラーがコンストラクタで IAmazonS3 または IAmazonDynamoDB のいずれかをパラメータとして使用している場合、依存関係インジェクションシステムはこれらのサービスを渡します。

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

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

    ...

}
```

他のアプリケーションパラメータの設定

> **Note**
>
> このトピックの情報は、.NET Framework に基づくプロジェクトに固有のものです。.NET Core に基づくプロジェクトでは、デフォルトでは App.config および Web.config ファイルは存在しません。

開いて .NET Framework のコンテンツを表示する

設定可能なアプリケーションパラメータは多数あります。

- AWSLogging
- AWSLogMetrics
- AWSRegion
- AWSResponseLogging
- AWS.DynamoDBContext.TableNamePrefix
- AWS.S3.UseSignatureVersion4
- AWSEndpointDefinition
- AWS サービス生成エンドポイント

これらのパラメータは、アプリケーションの App.config ファイルまたは Web.config ファイルで設定できます。 AWS SDK for .NET API を使用してこれらを設定することもできますが、アプリケーションの .config ファイルを使用することをお勧めします。ここでは、両方のアプローチについて説明します。

このトピックで後ほど説明する <aws> エレメントの使用方法の詳細については、「AWS SDK for .NETの設定ファイルリファレンス」を参照してください。

AWSLogging

SDK でイベントを記録する方法を設定します。例えば、推奨される方法としては <aws> 要素の子要素である <logging> 要素の使用があります。

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

または:

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

指定できる値は以下のとおりです。

None

イベントのログ記録を無効にします。これがデフォルトです。

`log4net`

log4net を使用してログを記録します。

`SystemDiagnostics`

`System.Diagnostics` クラスを使用してログを記録します。

コンマで区切ることで、`logTo` 属性に複数の値を設定できます。次の例では、`.config` ファイルを使用して `log4net` ログと `System.Diagnostics` ログの両方を設定します。

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

または:

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

または、 AWS SDK for .NET API を使用して、LoggingOptions 列挙の値を組み合わせて、AWSConfigs.Logging プロパティを設定します。

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

この設定の変更は、新しい AWS クライアントインスタンスに対してのみ有効になります。

AWSLogMetrics

SDK でパフォーマンスメトリクスを記録するかどうかを指定します。`.config` ファイルでメトリクスのログを設定するには、`<logging>` 要素の `logMetrics` 属性値を設定します。これは、`<aws>` 要素の子要素です。

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

または、`<appSettings>` セクションで `AWSLogMetrics` キーを設定します。

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

または、 AWS SDK for .NET API でメトリクスのログ記録を設定するには、AWSConfigs.LogMetrics プロパティを設定します。

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

この設定では、すべてのクライアント/設定のデフォルトの LogMetrics プロパティを設定します。この設定の変更は、新しい AWS クライアントインスタンスに対してのみ有効になります。

AWSRegion

AWS リージョンを明示的に指定していないクライアントのデフォルトリージョンを設定します。.config ファイルでリージョンを設定する推奨される方法は、aws 要素の region 属性値を設定することです。

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

または、<appSettings> セクションで AWSRegion キーを設定します。

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

または、 AWS SDK for .NET API を使用してリージョンを設定するには、AWSConfigs.AWSRegion プロパティを設定します。

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

特定のリージョンの AWS クライアントの作成の詳細については、AWS 「リージョンの選択」を参照してください。この設定の変更は、新しい AWS クライアントインスタンスに対してのみ有効になります。

AWSResponseLogging

SDK がサービス応答を記録するタイミングを設定します。指定できる値は以下のとおりです。

Never

サービス応答を記録しません。これがデフォルトです。

Always

常にサービス応答を記録します。

OnError

エラーが発生したときのみサービス応答を記録します。

.config ファイルでサービス ログを設定する推奨される方法は、<logging> 要素の logResponses 属性値を設定することです。これは、<aws> 要素の子要素です。

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

または、<appSettings> セクションで AWSResponseLogging キーを設定します。

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

または、 AWS SDK for .NET API でサービスログ記録を設定するには、AWSConfigs.ResponseLogging プロパティを ResponseLoggingOption 列挙のいずれかの値に設定します。

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

この設定に対する変更はすぐに反映されます。

AWS.DynamoDBContext.TableNamePrefix

手動で設定しなかった場合に DynamoDBContext で使用されるデフォルトの TableNamePrefix を設定します。

.config ファイルでテーブル名プレフィックスを設定する際の推奨される方法は、<dynamoDBContext> 要素の tableNamePrefix 属性値を設定することです。この要素は <dynamoDB> 要素の子要素であり、これ自体は <aws> 要素の子要素です。

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

または、<appSettings> セクションで AWS.DynamoDBContext.TableNamePrefix キーを設定します。

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

または、 AWS SDK for .NET API でテーブル名のプレフィックスを設定するには、AWSConfigs.DynamoDBContextTableNamePrefix プロパティを設定します。

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

この設定への変更は、DynamoDBContextConfig および DynamoDBContext の新しく構築されたインスタンスのみに有効です。

AWS.S3.UseSignatureVersion4

Amazon S3 クライアントがリクエストへの署名で署名バージョン 4 を使用するかどうかを設定します。

Amazon S3 の署名バージョン 4 での署名を .config ファイルで設定する際に推奨される方法は、<s3> 要素の useSignatureVersion4 属性を設定することです。これは、<aws> 要素の子要素です。

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

または、AWS.S3.UseSignatureVersion4 セクションで true キーを <appSettings> に設定します。

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

または、 AWS SDK for .NET API で署名バージョン 4 の署名を設定するには、AWSConfigs.S3UseSignatureVersion4 プロパティを に設定しますtrue。

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

デフォルトでは、この設定は false ですが、一部のケースまたは一部のリージョンではデフォルトで署名バージョン 4 が使用される場合があります。設定が true の場合、すべてのリクエストに署名バージョン 4 が使用されます。この設定の変更は新しい Amazon S3 クライアントインスタンスに対してのみ有効です。

AWSEndpointDefinition

SDK がリージョンとエンドポイントを定義するカスタム設定ファイルを使用するかどうかを設定します。

.config ファイルでエンドポイント定義ファイルを設定するには、<aws> 要素の endpointDefinition 属性値を設定することをお勧めします。

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

または、<appSettings> セクションで AWSEndpointDefinition キーを設定できます。

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

または、 AWS SDK for .NET API を使用してエンドポイント定義ファイルを設定するには、AWSConfigs.EndpointDefinition プロパティを設定します。

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

ファイル名が指定されていない場合、カスタム設定ファイルは使用されません。この設定の変更は、新しい AWS クライアントインスタンスに対してのみ有効になります。endpoints.json ファイルは から入手できますhttps://github.com/aws/aws-sdk-net/blob/main/sdk/src/Core/endpoints.json。

AWS サービス生成エンドポイント

一部の AWS サービスは、リージョンエンドポイントを使用する代わりに独自のエンドポイントを生成します。これらのサービスのクライアントは、そのサービスおよびリソースに固有のサービス URL を使用します。これらのサービスの 2 つの例は、Amazon CloudSearch と です AWS IoT。次の例は、これらのサービスのエンドポイントを取得する方法を示しています。

Amazon CloudSearch エンドポイントの例

Amazon CloudSearch クライアントは、Amazon CloudSearch 設定サービスにアクセスするために使用されます。Amazon CloudSearch 設定サービスを使用して、検索ドメインを作成、設定、管理します。検索ドメインを作成するには、CreateDomainRequest オブジェクトを作成し、DomainName プロパティを指定します。リクエストオブジェクトを使用して AmazonCloudSearchClient オブジェクトを作成します。CreateDomain メソッドを呼び出します。呼び出しから返される CreateDomainResponse オブジェクトには、DocService および SearchService エンドポイントの両方を持つ DomainStatus プロパティが含まれます。AmazonCloudSearchDomainConfig オブジェクトを作成し、それを使用して AmazonCloudSearchDomainClient クラスの DocService および SearchService インスタンスを初期化します。

```
// 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 エンドポイントの例

のエンドポイントを取得するには AWS IoT、AmazonIoTClient オブジェクトを作成し、DescribeEndPoint メソッドを呼び出します。返される DescribeEndPointResponse オブジェクトには、EndpointAddress が含まれます。AmazonIotDataConfig オブジェクトを作成し、ServiceURL プロパティを設定します。次に、このオブジェクトを使用して AmazonIotDataClient クラスをインスタンス化します。

```
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");
}
```

の設定ファイルリファレンス AWS SDK for .NET

> **Note**
>
> このトピックの情報は、.NET Framework に基づくプロジェクトに固有のものです。.NET Core に基づくプロジェクトでは、デフォルトでは App.config および Web.config ファイルは存在しません。

開いて .NET Framework のコンテンツを表示する

.NET プロジェクトの App.configまたは Web.config ファイルを使用して、 AWS 認証情報、ログ記録オプション、 AWS サービスエンドポイント、 AWS リージョンなどの AWS 設定と、Amazon DynamoDB、Amazon EC2、Amazon S3 などの AWS サービスの一部の設定を指定できます。以下では、適切な形式の App.config ファイルまたは Web.config ファイルを使用してこれらの設定を指定する方法について説明します。

> **Note**
>
> App.config または Web.config ファイルで <appSettings>要素を引き続き使用して AWS 設定を指定できますが、このトピックで後述する 要素<configSections>と<aws>要素を使用することをお勧めします。<appSettings> 要素の詳細については、AWS SDK for .NET 「アプリケーションの設定」の<appSettings>要素の例を参照してください。

> **Note**
>
> コードファイルで引き続き次のAWSConfigsクラスプロパティを使用して AWS 設定を指定できますが、次のプロパティは廃止され、今後のリリースではサポートされない場合があります。
>
> - DynamoDBContextTableNamePrefix
> - EC2UseSignatureVersion4
> - LoggingOptions
> - LogMetrics
> - ResponseLoggingOption
> - S3UseSignatureVersion4
>
> 一般的に、コードファイルでAWSConfigsクラスプロパティを使用して AWS 設定を指定する代わりに、このトピックで後述するように、 App.configまたは Web.config ファイルの <configSections>および <aws>要素を使用して AWS 設定を指定することをお勧めします。上記のプロパティの詳細については、AWS SDK for .NET 「アプリケーションの設定」のAWSConfigsコード例を参照してください。

トピック

- AWS 設定セクションの宣言
- 使用できる要素
- 要素のリファレンス

AWS 設定セクションの宣言

<aws> 要素内から App.configまたは Web.config ファイルで AWS 設定を指定します。<aws> 要素の使用を始める前に、次の例で示すように <section> 要素 (<configSections> 要素の子要素) を作成し、その name 属性を aws に、type 属性を Amazon.AWSSection, AWSSDK.Core に設定する必要があります。

```
<?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 のエディタでは、<aws> 要素またはその子要素の自動コード補完機能（IntelliSense）は提供されていません。

<aws> 要素を正しくフォーマットするには、Amazon.AWSConfigs.GenerateConfigTemplate メソッドを呼び出してください。このメソッドは <aws> 要素の正規バージョンを適切な文字列として出力するので、それを利用できます。以下のセクションでは、<aws> 要素の属性と子要素について説明します。

使用できる要素

AWS 設定セクションで使用できる要素の間の論理的な関係を次に示します。このリストの最新バージョンは Amazon.AWSConfigs.GenerateConfigTemplate メソッドを呼び出すことによって生成できます。このメソッドは、<aws> 要素の正規バージョンをそのまま使用できる文字列として出力します。

```
<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>
```

要素のリファレンス

以下は、 AWS 設定セクションで許可される要素のリストです。各要素について、使用できる属性と親子要素が示されています。

トピック

- alias
- aws
- dynamoDB
- dynamoDBContext
- Ec2
- ログ記録
- マップ
- property
- proxy
- s3

alias

<alias> 要素は、1 つ以上のテーブル間マッピングのコレクションに含まれる、タイプに対して設定されているものとは異なるテーブルを指定する単一の項目を表します この要素は、 AWS SDK for .NETの Amazon.AWSConfigs.DynamoDBConfig.Context.TableAliases プロパティから Amazon.Util.TableAlias クラスのインスタンスにマッピングされます。テーブル名のプレフィックスを適用する前に再マッピングが行われます。

この要素は次の属性を含むことができます:

fromTable

マップ元テーブルからマップ先テーブルへのマッピングのマップ元テーブル部分です。この属性は、 AWS SDK for .NETの Amazon.Util.TableAlias.FromTable プロパティにマッピングされます。

toTable

マップ元テーブルからマップ先テーブルへのマッピングのマップ先テーブル部分です この属性は、 AWS SDK for .NETの Amazon.Util.TableAlias.ToTable プロパティにマッピングされます。

<alias> 要素の親は、<tableAliases> 要素です。

<alias> 要素には子要素は含まれません。

次に <alias> 要素の使用例を示します。

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

aws

<aws> 要素は、 AWS 設定セクションの最上位要素を表します。この要素は次の属性を含むことができます:

endpointDefinition

使用する AWS リージョンとエンドポイントを定義するカスタム設定ファイルへの絶対パス。この属性は、 AWS SDK for .NETの Amazon.AWSConfigs.EndpointDefinition プロパティにマッピングされます。

profileName

サービス呼び出しを行うために使用される、保存された AWS 認証情報のプロファイル名。この属性は、 AWS SDK for .NETの Amazon.AWSConfigs.AWSProfileName プロパティにマッピングされます。

profilesLocation

他の AWS SDKs と共有されている認証情報ファイルの場所への絶対パス。デフォルトでは、認証情報ファイルは現在のユーザーのホームディレクトリにある .aws ディレクトリに格納されます。この属性は、 AWS SDK for .NETの Amazon.AWSConfigs.AWSProfilesLocation プロパティにマッピングされます。

region

AWS リージョンを明示的に指定していないクライアントのデフォルトのリージョン ID。この属性は、 AWS SDK for .NETの Amazon.AWSConfigs.AWSRegion プロパティにマッピングされます。

<aws> 要素に親要素はありません。

<aws> 要素は次の子要素を含むことができます。

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

次に <aws> 要素の使用例を示します。

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

dynamoDB

<dynamoDB> 要素は、Amazon DynamoDB の設定のコレクションを表します。この要素は、conversionSchema 属性を含むことができます。この属性は、.NET と DynamoDB オブジェクトの間の変換に使用するバージョンを表します。使用できる値は、V1 および V2 です。この属性は、AWS SDK for .NETの Amazon.DynamoDBv2.DynamoDBEntryConversion クラスにマッピングされます。詳細については、「DynamoDB Series - Conversion Schemas」を参照してください。

<dynamoDB> 要素の親は、<aws> 要素です。

<dynamoDB> 要素は、<dynamoDBContext> 子要素を含むことができます。

次に <dynamoDB> 要素の使用例を示します。

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

dynamoDBContext

<dynamoDBContext> 要素は、Amazon DynamoDB コンテキスト固有の設定のコレクションを表します。この要素は tableNamePrefix 属性を含むことができます。この属性は、テーブル名プレフィックスが手動で設定されていない場合に DynamoDB コンテキスト

が使用するデフォルトのプレフィックスを表します。この属性は、 AWS SDK for .NETの Amazon.AWSConfigs.DynamoDBConfig.Context.TableNamePrefix プロパティから Amazon.Util.DynamoDBContextConfig.TableNamePrefix プロパティにマッピングされます。詳細については、「Enhancements to the DynamoDB SDK」を参照してください。

<dynamoDBContext> 要素の親は、<dynamoDB> 要素です。

<dynamoDBContext> 要素は次の子要素を含むことができます。

- <alias> (1 つまたは複数のインスタンス)
- <map> (1 つまたは複数のインスタンス)

次に <dynamoDBContext> 要素の使用例を示します。

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

Ec2

<ec2> 要素は、Amazon EC2 の設定のコレクションを表します。この要素は、useSignatureVersion4 属性を含むことができます。この属性は、署名バージョン 4 の署名をすべての要求に対して使用すること (true)、または署名バージョン 4 の署名をすべての要求に対して使用しないこと (false、デフォルト) を指定します。この属性は、 AWS SDK for .NETの Amazon.AWSConfigs.EC2Config.UseSignatureVersion4 プロパティから Amazon.Util.EC2Config.UseSignatureVersion4 プロパティにマッピングされます。

<ec2> 要素の親は、要素です。

<ec2> 要素には子要素は含まれません。

次に <ec2> 要素の使用例を示します。

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

ログ記録

<logging> 要素は、応答ログおよびパフォーマンスメトリクスログの設定のコレクションを表します。この要素は次の属性を含むことができます:

logMetrics

パフォーマンスメトリクスをすべてのクライアントおよび設定に対して記録するか (true)、または記録しないか (false) を示します。この属性は、 AWS SDK for .NETの `Amazon.AWSConfigs.LoggingConfig.LogMetrics` プロパティから `Amazon.Util.LoggingConfig.LogMetrics` プロパティにマッピングされます。

logMetricsCustomFormatter

ログ記録メトリクスのカスタムフォーマッターのデータ型およびアセンブリ名です この属性は、AWS SDK for .NETの `Amazon.AWSConfigs.LoggingConfig.LogMetricsCustomFormatter` プロパティから `Amazon.Util.LoggingConfig.LogMetricsCustomFormatter` プロパティにマッピングされます。

logMetricsFormat

ログ記録メトリクスを示す形式です (AWS SDK for .NETの `Amazon.AWSConfigs.LoggingConfig.LogMetricsFormat` プロパティから `Amazon.Util.LoggingConfig.LogMetricsFormat` プロパティにマッピングされます)。

使用できる値は次のとおりです。

JSON

JSON 形式を使用します。

Standard

デフォルトの形式を使用します。

logResponses

サービス応答をいつログに記録するかを示します (AWS SDK for .NETの `Amazon.AWSConfigs.LoggingConfig.LogResponses` プロパティから `Amazon.Util.LoggingConfig.LogResponses` プロパティにマッピングされます)。

使用できる値は次のとおりです。

Always

常にサービス応答を記録します。

Never

サービス応答を記録しません。

OnError

エラーがあるときにのみサービス応答を記録します。

logTo

ログ記録先 (の LogToプロパティから Amazon.AWSConfigs.LoggingConfig.LogToプロパティにマップします AWS SDK for .NET) 。

使用できる値は次のとおりです。

Log4Net

log4net にログを記録します。

None

ログを無効にします。

SystemDiagnostics

System.Diagnostics にログを記録します。

<logging> 要素の親は、<aws> 要素です。

<logging> 要素には子要素は含まれません。

次に <logging> 要素の使用例を示します。

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

マップ

<map> 要素は、.NET の型から DynamoDB のテーブルへのマッピングのコレクション内の 1 つの項目を表します (AWS SDK for .NETの Amazon.AWSConfigs.DynamoDBConfig.Context.TypeMappings プロパティから TypeMapping クラスのインスタンスにマッピングされます)。詳細については、「Enhancements to the DynamoDB SDK」を参照してください。

この要素は次の属性を含むことができます:

`targetTable`

マッピングが適用される DynamoDB のテーブルです。この属性は、 AWS SDK for .NETの `Amazon.Util.TypeMapping.TargetTable` プロパティにマッピングされます。

`type`

マッピングが適用される型とアセンブリ名です この属性は、 AWS SDK for .NETの `Amazon.Util.TypeMapping.Type` プロパティにマッピングされます。

`<map>` 要素の親は、`<dynamoDBContext>` 要素です。

`<map>` 要素は、`<property>` 子要素の 1 つまたは複数のインスタンスを含むことができます。

次に `<map>` 要素の使用例を示します。

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

property

`<property>` 要素は DynamoDB のプロパティを表します。(この要素は、 の `AddProperty`メソッドから Amazon.Util.PropertyConfig クラスのインスタンスにマッピングされます AWS SDK for .NET)。詳細については、DynamoDB SDK および DynamoDB 属性の機能強化DynamoDB」を参照してください。

この要素は次の属性を含むことができます:

`attribute`

範囲キーの名前など、プロパティの属性の名前です。この属性は、 AWS SDK for .NETの `Amazon.Util.PropertyConfig.Attribute` プロパティにマッピングされます。

`converter`

このプロパティに使用する必要があるコンバーターの種類です この属性は、 AWS SDK for .NET の `Amazon.Util.PropertyConfig.Converter` プロパティにマッピングされます。

ignore

関連付けられているプロパティを無視する必要があるか (true)、またはないか (false) を示します。この属性は、 AWS SDK for .NETの Amazon.Util.PropertyConfig.Ignore プロパティにマッピングされます。

name

プロパティの名前。この属性は、 AWS SDK for .NETの Amazon.Util.PropertyConfig.Name プロパティにマッピングされます。

version

このプロパティが項目のバージョン番号を格納する必要があるか (true)、またはないか (false) を示します。この属性は、 AWS SDK for .NETの Amazon.Util.PropertyConfig.Version プロパティにマッピングされます。

<property> 要素の親は、<map> 要素です。

<property> 要素には子要素は含まれません。

次に <property> 要素の使用例を示します。

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

proxy

<proxy> 要素は、 AWS SDK for .NET で使用するプロキシの設定を表します。この要素は次の属性を含むことができます:

ホスト

プロキシサーバーのホスト名または IP アドレスです この属性は、 AWS SDK for .NETの Amazon.AWSConfigs.ProxyConfig.Host プロパティから Amazon.Util.ProxyConfig.Host プロパティにマッピングされます。

password

プロキシサーバーで認証するためのパスワードです。この属性は、 AWS SDK for .NETの Amazon.AWSConfigs.ProxyConfig.Password プロパティから Amazon.Util.ProxyConfig.Password プロパティにマッピングされます。

port

プロキシのポート番号です。この属性は、 AWS SDK for .NET の Amazon.AWSConfigs.ProxyConfig.Port プロパティから Amazon.Util.ProxyConfig.Port プロパティにマッピングされます。

username

プロキシサーバーで認証するユーザー名です。この属性は、 AWS SDK for .NETの Amazon.AWSConfigs.ProxyConfig.Username プロパティから Amazon.Util.ProxyConfig.Username プロパティにマッピングされます。

<proxy> 要素の親は、<aws> 要素です。

<proxy> 要素には子要素は含まれません。

次に <proxy> 要素の使用例を示します。

```
<proxy
  host="192.0.2.0"
  port="1234"
  username="My-Username-Here"
  password="My-Password-Here" />
```

s3

<s3> 要素は、Amazon S3 の設定のコレクションを表します。この要素は、useSignatureVersion4 属性を含むことができます。この属性は、署名バージョン 4 の署名をすべての要求に対して使用すること (true)、または署名バージョン 4 の署名をすべての要求に対して使用しないこと (false、デフォルト) を指定します。この属性は、 AWS SDK for .NETの Amazon.AWSConfigs.S3Config.UseSignatureVersion4 プロパティにマッピングされます。

<s3> 要素の親は、<aws> 要素です。

<s3> 要素には子要素は含まれません。

次に <s3> 要素の使用例を示します。

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

レガシー認証情報の使用

このセクションのトピックでは、 AWS IAM アイデンティティセンターを使用せずに長期または短期認証情報を使用する方法について説明します。

> **Warning**
>
> セキュリティリスクを避けるため、専用ソフトウェアを開発するときや実際のデータを扱うときは、IAM ユーザーを認証に使用しないでください。代わりに、AWS IAM アイデンティティセンター などの ID プロバイダーとのフェデレーションを使用してください。

> **Note**
>
> このトピックの情報は、短期または長期認証情報を手動で取得および管理する必要がある場合を対象としています。短期認証情報と長期認証情報に関する追加情報については、「AWS SDK およびツールリファレンスガイド」の「その他の認証方法」を参照してください。セキュリティのベストプラクティスについては、 AWS IAM アイデンティティセンター「」の説明に従って を使用しますによる認証 AWS。

認証情報に関する重要な警告とガイダンス

認証情報に関する警告

- アカウントのルート認証情報を使用して AWS リソースにアクセスしないでください。これらの認証情報は無制限のアカウントアクセスを提供し、取り消すのが困難です。
- アプリケーションファイルにリテラルアクセスキーや認証情報を配置しないでください。これを行うと、パブリックリポジトリにプロジェクトをアップロードするなど、誤って認証情報が公開されるリスクが発生します。
- プロジェクト領域に認証情報を含むファイルを含めないでください。
- 共有 AWS `credentials`ファイルに保存されている認証情報はすべてプレーンテキストで保存されることに注意してください。

認証情報を安全に管理するための追加のガイダンス

AWS 認証情報を安全に管理する方法の一般的な説明については、「」のAWS 「セキュリティ認証情報AWS 全般のリファレンス」および「IAM ユーザーガイド」の「セキュリティのベストプラクティスとユースケース」を参照してください。これらの説明に加えて、以下の点を考慮してください。

- IAM Identity Center のユーザーなど、追加ユーザーを作成し、 AWS ルートユーザー認証情報を使用する代わりにそのユーザー認証情報を使用します。他のユーザーの認証情報は、必要に応じて取り消すこともできますが、本質的に一時的なものです。さらに、各ユーザーに対して、特定のリソースとアクションにアクセスするためのポリシーを適用できます。これにより、最小特権のアクセス許可になります。

- Amazon Elastic Container Service (Amazon ECS) タスクで、タスク用の IAM ロールを使用します。

- Amazon EC2 インスタンスで実行中のアプリケーションに対して、IAM ロールを使用します。

- 組織外部のユーザーが利用できるアプリケーションでは、一時的な認証情報または環境変数を使用します。

トピック

- 共有 AWS 認証情報ファイルの使用
- SDK ストアの使用 (Windows のみ)

共有 AWS 認証情報ファイルの使用

(認証情報に関する重要な警告とガイダンスを必ずご確認ください)

アプリケーションに認証情報を提供する方法の 1 つは、共有 AWS 認証情報ファイルにプロファイルを作成し、そのプロファイルに認証情報を保存することです。このファイルは、他の AWS SDKsで使用できます。また、、AWS Tools for Windows PowerShell、および Visual StudioAWS CLI、JetBrains、VS Code の AWS ツールキットでも使用できます。

> **Warning**
>
> セキュリティリスクを避けるため、専用ソフトウェアを開発するときや実際のデータを扱うときは、IAM ユーザーを認証に使用しないでください。代わりに、AWS IAM アイデンティティセンター などの ID プロバイダーとのフェデレーションを使用してください。

> **Note**
>
> このトピックの情報は、短期または長期認証情報を手動で取得および管理する必要がある場合を対象としています。短期認証情報と長期認証情報に関する追加情報については、「AWS SDK およびツールリファレンスガイド」の「その他の認証方法」を参照してください。セキュリティのベストプラクティスについては、 AWS IAM アイデンティティセンター「」の説明に従って を使用しますによる認証 AWS。

一般的な情報

デフォルトでは、共有 AWS 認証情報ファイルはホーム.awsディレクトリ内の ディレクトリにありcredentials、 (Linux または macOS) または ~/.aws/credentials (%USERPROFILE%\.aws\credentialsWindows) という名前が付けられます。別の保存場所に関する詳細については、AWS SDK とツールのリファレンスガイドの「共有ファイルの場所」を参照してください。また、「アプリケーションでの認証情報とプロファイルへのアクセス」も参照してください。

共有 AWS 認証情報ファイルはプレーンテキストファイルであり、特定の形式に従います。 AWS 認証情報ファイルの形式については、 SDK およびツールリファレンスガイドの「認証情報ファイルの形式」を参照してください。 AWS SDKs

共有 AWS 認証情報ファイルのプロファイルは、いくつかの方法で管理できます。

- 任意のテキストエディタを使用して、共有 AWS 認証情報ファイルを作成および更新します。
- このトピックで後述するように、 AWS SDK for .NET API の Amazon.Runtime.CredentialManagement 名前空間を使用します。
- および Visual Studio、JetBrains、VS Code のAWS Tools for PowerShell AWS ツールキットのコマンドと手順を使用します。 https://docs.aws.amazon.com/toolkit-for-vscode/latest/userguide/setup-credentials.html
- AWS CLI コマンドを使用します (例えば `aws configure set aws_access_key_id` および `aws configure set aws_secret_access_key`)。

プロファイル管理の例

以下のセクションでは、共有 AWS 認証情報ファイルのプロファイルの例を示します。いくつかの例では、前述の認証情報管理方法のいずれかを使用して取得できる結果が示されています。その他の例では、特定のメソッドの使用方法を示しています。

デフォルトのプロファイル

共有 AWS 認証情報ファイルには、ほとんどの場合、default という名前のプロファイルがあります。これは、他のプロファイルが定義されていない場合に が認証情報 AWS SDK for .NET を検索する場所です。

[default] プロファイルは通常、以下のようになっています。

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

プログラムでのプロファイルの作成

この例では、プロファイルを作成し、それをプログラムで共有 AWS 認証情報ファイルに保存する方法を示します。Amazon.Runtime.CredentialManagement 名前空間の次のクラスを使用します: CredentialProfileOptions、CredentialProfile、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**
>
> このようなコードは、通常、アプリケーションに含めるべきではありません。アプリケーションに組み込む場合は、プレーンテキストキーがコード内、ネットワーク経由、またはコンピュータのメモリ内で見えないように適切な予防措置を講じてください。

この例で作成されたプロファイルを以下に示します。

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

既存のプロファイルのプログラムでの更新

この例では、前の手順で作成したプロファイルをプログラムで更新する方法を示します。Amazon.Runtime.CredentialManagement 名前空間の次のクラスを使用します: CredentialProfile、SharedCredentialsFile。また、Amazon 名前空間の RegionEndpoint クラスも使用します。

```
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);
    }
}
```

更新されたプロファイルを以下に示します。

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

> **Note**
>
> AWS リージョンは、他の場所や他の方法を使用して設定することもできます。詳細については、「の AWS リージョンの設定 AWS SDK for .NET」を参照してください。

SDK ストアの使用 (Windows のみ)

(重要な警告とガイドラインを必ずご確認ください)

Windows では、SDK ストアは AWS SDK for .NET 、アプリケーションのプロファイルを作成し、暗号化された認証情報を保存するもう 1 つの場所です。`%USERPROFILE%\AppData\Local\AWSToolkit\RegisteredAccounts.json` にあります。開発時に、共有 AWS 認証情報ファイルの代わりとして SDK ストアを使用できます。

> **Warning**
>
> セキュリティリスクを避けるため、専用ソフトウェアを開発するときや実際のデータを扱うときは、IAM ユーザーを認証に使用しないでください。代わりに、AWS IAM アイデンティティセンター などの ID プロバイダーとのフェデレーションを使用してください。

> **Note**
>
> このトピックの情報は、短期または長期認証情報を手動で取得および管理する必要がある場合を対象としています。短期認証情報と長期認証情報に関する追加情報については、「AWS SDK およびツールリファレンスガイド」の「その他の認証方法」を参照してください。セキュリティのベストプラクティスについては、 AWS IAM アイデンティティセンター「」の説明に従って を使用しますによる認証 AWS。

一般的な情報

SDK ストアには次の利点があります。

- SDK ストア内の認証情報が暗号化された状態で、ユーザーのホームディレクトリに SDK ストアが配置されます。これにより、認証情報が誤って公開されるリスクが制限されます。
- SDK ストアは、AWS Tools for Windows PowerShell および AWS Toolkit for Visual Studio にも認証情報を提供します。

SDK ストアのプロファイルは、特定のホスト上の特定ユーザーに固有です。これらは他のホストや他のユーザーにコピーすることはできません。そのため、開発用マシンの SDK ストアにあるプロファイルを、他のホストや他のデベロッパーのマシンで再利用することはできません。また、本番稼働用のアプリケーションで SDK ストアのプロファイルを使用できないということでもあります。

SDK ストアのプロファイルは、以下に示すいくつかの方法で管理できます。

- AWS Toolkit for Visual Studio のグラフィカルユーザーインターフェイス (GUI) を使用します。
- このトピックで後述するように、 AWS SDK for .NET API の Amazon.Runtime.CredentialManagement 名前空間を使用します。
- AWS Tools for Windows PowerShell のコマンドを使用します (例えば `Set-AWSCredential` および `Remove-AWSCredentialProfile`)。

プロファイル管理の例

以下の例では、プログラムで SDK ストア内にプロファイルを作成して更新する方法を示します。

プログラムでのプロファイルの作成

この例では、プログラムを使用してプロファイルを作成し、SDK ストアに保存する方法を説明します。Amazon.Runtime.CredentialManagement 名前空間の次のクラスを使用します: CredentialProfileOptions、CredentialProfile、NetSDKCredentialsFile。

```
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**
>
> このようなコードは、通常、アプリケーションに含めるべきではありません。アプリケーションに含まれる場合は、プレーンテキストキーがコード内、ネットワーク経由、またはコンピュータのメモリ内で見えないように適切な予防措置を講じてください。

この例で作成されたプロファイルを以下に示します。

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

既存のプロファイルのプログラムでの更新

この例では、前の手順で作成したプロファイルをプログラムで更新する方法を示します。Amazon.Runtime.CredentialManagement 名前空間の次のクラスを使用します: CredentialProfile、NetSDKCredentialsFile。また、Amazon 名前空間の RegionEndpoint クラスも使用します。

```
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);
    }
}
```

更新されたプロファイルを以下に示します。

```
"[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**
>
> AWS リージョンは、他の場所や他の方法を使用して設定することもできます。詳細については、「の AWS リージョンの設定 AWS SDK for .NET」を参照してください。

の使用 AWS SDK for .NET

このセクションでは、アプリケーションの作成時に考慮する必要がある AWS SDK for .NET の機能について説明します。

開始する前に、環境をセットアップし、プロジェクトを設定したことを確認してください。

特定の AWS サービス用のソフトウェアの開発とコード例については、「」を参照してください AWS サービスの呼び出し。その他のコード例については、「SDK for .NET (v4) コード例」を参照してください。

トピック

- を使用した非同期プログラミング AWS SDK for .NET
- でページ分割された結果を使用する AWS SDK for .NET
- での HTTP 2 のサポート AWS SDK for .NET
- その他のツール

を使用した非同期プログラミング AWS SDK for .NET

は、非同期実装にタスクベースの非同期パターン (TAP) AWS SDK for .NET を使用します。TAP の詳細については、docs.microsoft.com の「タスクベースの非同期パターン (TAP)」を参照してください。

このトピックでは、 AWS サービスクライアントへの呼び出しで TAP を使用する方法の概要を説明します。

AWS SDK for .NET API の非同期メソッドは、 Task クラスまたは Task<TResult> クラスに基づくオペレーションです。Task class クラスおよび Task<TResult> class クラスの詳細については、docs.microsoft.com を参照してください。

これらの API メソッドがコードで呼び出される場合、次の例に示すように、async キーワードで宣言された関数内で呼び出される必要があります。

```
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;
}
```

前述のコードスニペットに示されているように、async 宣言の推奨スコープは Main 関数です。この async スコープを設定する場合、 AWS サービスクライアントに対するすべての呼び出しが非同期である必要があります。何らかの理由で Main を非同期として宣言できない場合は、次の例に示すように、Main 以外の関数で async キーワードを使用して、そこから API メソッドを呼び出します。

```
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;
}
```

このパターンを使用する場合、Main で必要とされる特殊な Task<> 構文に注意してください。また、レスポンスの **Result** メンバーを使用してデータを取得する必要があります。

AWS サービスクライアントへの非同期呼び出しの完全な例は、 シンプルなアプリケーションの作成セクション (シンプルなクロスプラットフォームアプリ および シンプルな Windows ベースのアプリ) および で確認できますガイド付きコードの例。

でページ分割された結果を使用する AWS SDK for .NET

一部の AWS サービスは、大量のデータを収集して保存し、 の API コールを使用して取得できます AWS SDK for .NET。取得するデータ量が 1 回の API コールとしては多すぎる場合、ページ分割を使用して、より管理しやすい大きさに結果を分割できます。

ページ分割を実行するには、SDK 内の多くのサービスクライアントのリクエストオブジェクトとレスポンスオブジェクトによって提供される 継続トークン (通常は `NextToken`) を使用します。また、このようなサービスクライアントの一部では、ページネーターも使用できます。

ページネーターを使用することで、ループ、状態変数、複数の API コールなどが影響を受ける場合がある継続トークンのオーバーヘッドを回避できます。ページネーターを使用すると、1 行のコード、すなわち `foreach` ループの宣言によって AWS からデータを取得できます。データを取得するために複数の API コールが必要な場合は、ページネーターがこれを処理します。

ページネーターの使用の可否

すべてのサービスでページネーターを使用できるわけではありません。サービスで特定の API 向けのページネーターを提供しているかどうかを判断する 1 つの方法は、AWS SDK for .NET API リファレンスでサービスクライアントクラスの定義を確認します。

たとえば、AmazonCloudWatchlogsClient クラスの定義を調べると、`Paginators` プロパティが見つかります。これは、Amazon CloudWatch Logs 用のページネーターを提供するプロパティです。

ページネーターを使用するメリット

ページネーターには、完全なレスポンスを表示するプロパティが含まれています。また、通常、主な結果であるレスポンスの最も興味深い部分へのアクセスを有効にする 1 つ以上のプロパティも含まれています。。

たとえば、前述の `AmazonCloudWatchLogsClient` では、`Paginator` オブジェクトに `Responses` プロパティと、API コールによる完全な DescribeLogGroupsResponse オブジェクトが含まれています。特に、この `Responses` プロパティには、ロググループのコレクションが含まれています。

また、ページネーターオブジェクトにも、`LogGroups` という名前の主な結果が 1 つ含まれています。このプロパティにはレスポンスのロググループ部分のみが格納されています。この主な結果が含まれていることで、多くの状況でコードを減らして簡素化できます。

同期と非同期のページ分割

ページネーターでは、ページ分割の同期と非同期の両方の仕組みを利用できます。同期ページ分割は、.NET Framework 4.7.2 (またはそれ以降) プロジェクトで使用できます。非同期ページ分割は、.NET Core プロジェクト (.NET Core 3.1、.NET 5 など) で使用できます。

非同期操作と.NET Core が推奨されているため、次の例では非同期ページ分割について説明します。同期ページ分割と .NET Framework 4.7.2 (またはそれ以降) を使用して同じタスクを実行する方法については、「」の例の後に示されています[ページネーターに関する追加の考慮事項](#)。

例

次の例は、 を使用してロググループのリスト AWS SDK for .NET を表示する方法を示しています。比較のために、この例では、ペジネーターを使用した方法と、使用しない方法の両方を示します。後で示す完全なコードを確認する前に、次のスニペットを検討してください。

ページネーターを使用せずに CloudWatch ロググループを取得する

```
      // 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));
```

ページネーターを使用して CloudWatch ロググループを取得する

```
      // 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);
      }
```

これら 2 つのスニペットの結果はまったく同じであるため、ページネータを使用する利点が明確になります。

> **Note**
>
> 完全なコードをビルドして実行する前に、環境とプロジェクトがセットアップされていることを確認してください。
> 非同期ページネーターでは `IAsyncEnumerable` インターフェイスが使用されるため、Microsoft.Bcl.AsyncInterfaces NuGet パッケージが必要となる場合もあります。

コード全文

このセクションでは、例に関連する参考資料とコードの全文を示します。

SDK リファレンス

NuGet パッケージ:

- AWSSDK.CloudWatch

プログラミング要素:

- 名前空間 Amazon.CloudWatch

  クラス AmazonCloudWatchLogsClient
- 名前空間 Amazon.CloudWatchLogs.Model

  クラス DescribeLogGroupsRequest

  クラス DescribeLogGroupsResponse

  クラス LogGroup

完全なコード

```
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}");
        }
      }
    }
  }
}
```

ページネーターに関する追加の考慮事項

- ページネーターは複数回使用できない

  コード内の複数の場所に特定の AWS ページネーターの結果が必要な場合は、ページネーターオブジェクトを複数回使用しないでください。代わりに、使用する必要があるたびに新しいページネーターを作成してください。この概念は、前の `DisplayLogGroupsWithPaginators` メソッドの例のコードに示されています。

- 同期ページ分割

  同期ページ分割は、.NET Framework 4.7.2 (またはそれ以降) プロジェクトで使用できます。

  これを確認するには、.NET Framework 4.7.2 (またはそれ以降) プロジェクトを作成し、前述のコードをコピーします。次に、次の例に示すように、2 つの `foreach` ページネータコールから `await` キーワードのみを削除します。

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

  プロジェクトをビルドして実行し、非同期ページ分割の結果と同じ結果が生成されることを確認します。

での HTTP 2 のサポート AWS SDK for .NET

一部の AWS サービスとオペレーションには HTTP 2 が必要です。例えば、Amazon Transcribe Streaming の双方向ストリーミングは HTTP 1.1 では使用できないため、代わりに HTTP 2 が必要です。バージョン 4 で HTTP 2 のサポート AWS SDK for .NET が追加され、これらのオペレーションをアプリケーションで使用できるようになりました。双方向 HTTP 2 オペレーションの場合、ストリームを受信するときの SDK の動作は、HTTP 1.1 の動作と似ています。つまり、SDK を使用するアプリケーションがサービスにイベントを送信すると、リクエストには開発者によって割り当てられたパブリッシャーが割り当てられます。

この動作を実際に確認するには、Amazon Transcribe Streaming の次の例を検討してください。Amazon.TranscribeStreaming および Amazon.TranscribeStreaming.Model 名前空間を使用します。

この例では、開発者は .NET であるコールバック関数を使用して StartStreamTranscriptionRequest.AudioStreamPublisherプロパティを定義しますFunc。SDK は、 にFunc定義された を使用してAudioStreamPublisher、ユーザーのコードからイベントをプルし、ユーザーにストリーミングします。SDK は null を返すFuncまで を呼び出します。

このコードは、ファイルからの音声を Amazon Transcribe Streaming にストリーミングして処理する方法を示しています。

```
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**
>
> Amazon Bedrock の `InvokeModelWithBidirectionalStreamAsync`メソッド、Amazon.BedrockRuntime 名前空間など、一部の双方向 HTTP 2 オペレーションは、一部のイベントが発行されるまで、サービスクライアントでの最初の呼び出しからのレスポンスを返しません。この動作により、アプリケーションがブロックされる可能性があります。この状況を回避するには、パブリッシャーにイベントを提供するアプリケーションコードを分離し、サービスクライアントでオペレーションを呼び出すスレッドとは異なるスレッドで実行します。

追加の考慮事項

- AWS SDK for .NET HTTP 2 のサポートは、.NET 8 以降をターゲットとするバージョンでのみ使用できます。.NET Framework をターゲットとするバージョンでは利用できません。
- 詳細については、aws-sdk-net GitHub リポジトリの「PR 3730」を参照してください。

その他のツール

.NET アプリケーションの開発、デプロイ、および保守の作業を容易にするために使用できる追加のツールを次に示します。

AWS デプロイツール

開発マシンでクラウドネイティブの .NET Core アプリケーションを開発したら、.NET CLI の AWS デプロイツールを使用して、アプリケーションを に簡単にデプロイできます AWS。

詳細については、「アプリケーションを にデプロイする AWS」を参照してください。

AWS .NET のメッセージ処理フレームワーク

Amazon SQS、Amazon SNS、Amazon EventBridge などのサービスを使用している場合は、AWS Message Processing Framework for .NET を利用できる場合があります。詳細については、「AWS .NET のメッセージ処理フレームワーク」を参照してください。

.NET Aspire との統合

.NET Aspire との統合を活用して、内部の開発ループを改善できます。詳細については、「での .NET Aspire AWS との統合 AWS SDK for .NET」を参照してください。

AWS SDK for .NETによる高度な認証と認可

このセクションのトピックでは、 AWS SDK for .NET アプリケーションでの認証と認可の高度な手法について説明します。

トピック

- を使用したシングルサインオン AWS SDK for .NET

を使用したシングルサインオン AWS SDK for .NET

AWS IAM アイデンティティセンター は、すべての AWS アカウント およびクラウドアプリケーションへの SSO アクセスを一元管理することを容易にするクラウドベースのシングルサインオン (SSO) サービスです。詳細については、IAM Identity Center ユーザーガイドを参照してください。

SDK が IAM Identity Centerとどのように相互作用するのかがよくわからない場合は、以下の情報を参照してください。

高レベルインタラクションステップ

高レベルでは、SDK は次のパターンと同様の方法で IAM Identity Centerと相互作用します。

1. IAM Identity Centerは通常 IAM Identity Centerコンソールを使用して設定され、SSO ユーザーは参加するよう招待されます。
2. ユーザーのコンピュータの共有ファイルが AWS `config`SSO 情報で更新されます。
3. ユーザーは IAM Identity Center を通じてサインインし、設定された AWS Identity and Access Management (IAM) アクセス許可の短期認証情報が与えられます。このサインインは、 などの非 SDK ツールを通じて開始することも AWS CLI、.NET アプリケーションを通じてプログラムで開始することもできます。
4. ユーザーは作業を続行します。SSO を使用している他のアプリケーションを実行する場合、アプリケーションを開くために再度サインインする必要はありません。

このトピックの残りの部分では、設定と AWS IAM アイデンティティセンターの使用に関する参考情報を提供します。による認証 AWSでの基本的な SSO 設定よりも補足的な情報やより高度な情報を提供します。SSO を初めて使用する場合は AWS、まずそのトピックで基本的な情報を確認し、次に以下のチュートリアルで SSO の実際の動作を確認することをお勧めします。

- チュートリアル: .NET アプリケーションのみ

- チュートリアル: AWS CLI および .NET アプリケーション

このトピックには、次のセクションが含まれています。

- 前提条件
- SSO プロファイルのセットアップ
- SSO トークンの生成と使用
- 追加リソース
- チュートリアル

前提条件

IAM Identity Center を使用する前に、ID ソースの選択や関連する AWS アカウント およびアプリケーションの設定など、特定のタスクを実行する必要があります。詳細については、以下を参照してください。

- IAM Identity Center の設定の詳細については、「IAM Identigy Center ユーザーガイド」の「Getting Started」 (使用開始) を参照してください。
- 具体的なタスクの例については、このトピックの最後にあるチュートリアルのリストを参照してください。ただし、チュートリアルを試す前に、このトピックの情報を必ず確認してください。

SSO プロファイルのセットアップ

関連する で IAM Identity Center を設定したら AWS アカウント、SSO の名前付きプロファイルをユーザーの共有 AWS `config`ファイルに追加する必要があります。このプロファイルは、AWS アクセス ポータルに接続するために使用され、ユーザーに対して構成された IAM 権限の短期認証情報を返します。

共有 `config` ファイルの名前は通常、Windows では `%USERPROFILE%\.aws\config`、Linux および macOS では `~/.aws/config` です。任意のテキストエディタを使用して SSO 用の新しいプロファイルを追加できます。または、 `aws configure sso` コマンドを使用できます。このコマンドの詳細については、「AWS Command Line Interface ユーザーガイド」の「IAM Identity Center を使用するための AWS CLI の設定」を参照してください。

新しいプロファイルは、次のようになります。

```
[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
```

新しいプロファイルの設定は以下で定義されています。最初の 2 つの設定は、 AWS アクセスポータルを定義します。他の 2 つの設定はペアになっており、これらを組み合わせてユーザーに設定された権限を定義します。4つの設定すべてが必須です。

sso_start_url

組織のAWS アクセスポータルを指す URL この値を確認するには、IAM Identity Center コンソールを開いて、[設定]を選択し、[ポータル URL] を探します。

sso_region

アクセスポータルホスト AWS リージョン を含む 。これは IAM Identity Centerを有効にしたときに選択されたリージョンです。他のタスクに使用するリージョンとは異なる場合があります。

AWS リージョン とそのコードの完全なリストについては、『』の「リージョンエンドポイント」を参照してくださいAmazon Web Services 全般のリファレンス。

sso_account_id

AWS Organizations サービスを通じて AWS アカウント 追加された の ID。使用可能なアカウントのリストを確認するには、IAM Identity Center コンソール に移動して AWS アカウントページを開きます。この設定で選択するアカウント ID は、次に示す `sso_role_name` 設定に指定する予定の値に対応します。

sso_role_name

IAM Identity Centerの権限セットの名前。この権限セットは、IAM Identity Center を通じてユーザーに付与される権限を定義します。

以下の手順は、この設定の値を確認する方法の 1 つです。

1. IAM Identity Centerコンソールに移動し、AWS アカウントページを開きます。
2. 詳細を表示するアカウントを選択します。選択するアカウントは、SSO 権限を付与したい SSO ユーザーまたはグループを含むアカウントになります。
3. アカウントに割り当てられているユーザーとグループのリストを見て、対象のユーザーまたはグループを探します。`sso_role_name`設定で指定する権限セットは、このユーザーまたはグループに関連付けられている権限セットの 1 つです。

この設定の値を指定するときは、Amazon リソースネーム (ARN) ではなく、権限セットの名前を使用してください。

権限セットには IAM ポリシーとカスタムアクセス権限ポリシーがアタッチされています。詳細については、「 IAM Identity Center ユーザーガイド」の「権限セット」を参照してください。

SSO トークンの生成と使用

SSO を使用するには、ユーザーはまず一時トークンを生成してから、そのトークンを使用して適切な AWS アプリケーションとリソースにアクセスする必要があります。.NET アプリケーションでは、次の方法を使用してこれらの一時トークンを生成して使用できます。

- 必要に応じて最初にトークンを生成し、そのトークンを使用する.NET アプリケーションを作成します。
- を使用してトークンを生成し AWS CLI 、そのトークンを .NET アプリケーションで使用します。

これらの方法については、以下のセクションで説明し、チュートリアルで実演しています。

> **Important**
>
> SSO 解決が機能するには、アプリケーションが次の NuGet パッケージを参照する必要があります。
>
> - `AWSSDK.SSO`
> - `AWSSDK.SSOOIDC`
>
> これらのパッケージを参照しないと、ランタイム例外が発生します。

.NETアプリケーションのみ

このセクションでは、必要に応じて一時的な SSO トークンを生成し、そのトークンを使用する .NET アプリケーションを作成する方法を説明します。このプロセスの詳細なチュートリアルについては、.NET アプリケーションのみを使用する SSO のチュートリアルを参照してください。

SSO トークンをプログラムで生成して使用する。

の使用に加えて AWS CLI、SSO トークンをプログラムで生成することもできます。

そのために、アプリケーションは SSO プロファイル用のAWSCredentialsオブジェクトを作成します。このオブジェクトは、一時的な認証情報があればそれをロードします。次に、アプリケーションはそのAWSCredentialsオブジェクトをSSOAWSCredentialsオブジェクトにキャストし、必要に応じてユーザーにサインイン情報の入力を求めるコールバックメソッドなど、いくつかの Options プロパティを設定する必要があります。

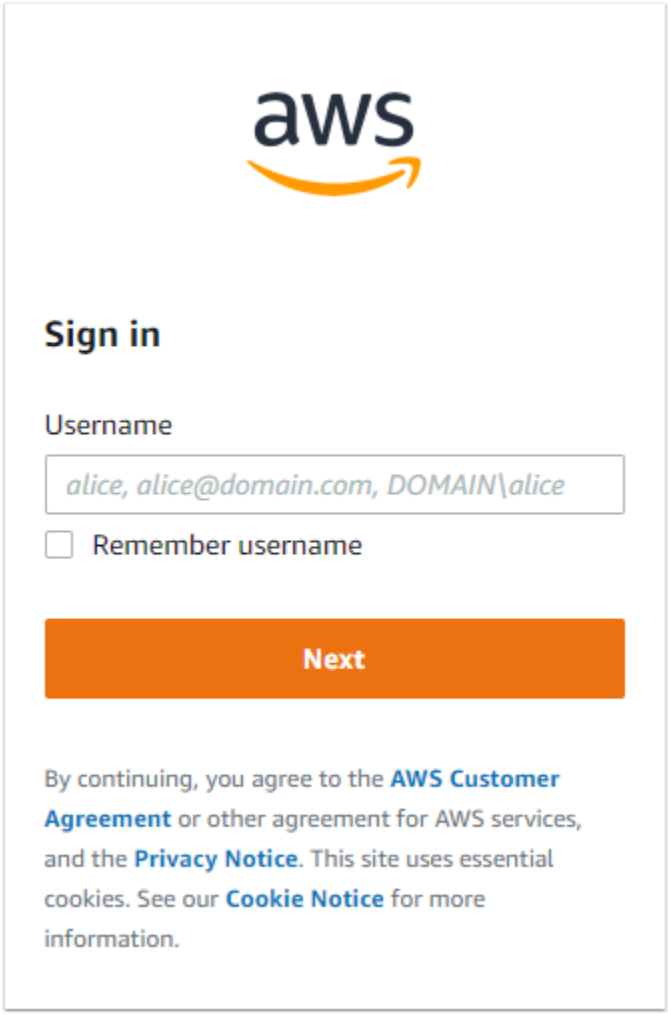
次のコードスニペットは、このメソッドを示しています。

> **Important**
>
> SSO 解決が機能するには、アプリケーションが次の NuGet パッケージを参照する必要があります。
>
> - AWSSDK.SSO
> - AWSSDK.SSOOIDC
>
> これらのパッケージを参照しないと、ランタイム例外が発生します。

```
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;
}
```

適切な SSO トークンがない場合は、既定のブラウザーウィンドウが起動し、適切なサインインページが開きます。たとえば、IAM Identity Center を ID ソースとして使用している場合、ユーザーには次のようなサインインページが表示されます。



> **Note**
>
> SSOAWSCredentials.Options.ClientNameに入力するテキスト文字列には、スペースを含めることはできません。文字列にスペースが含まれていると、ランタイム例外が発生します。

.NET アプリケーションのみを使用する SSO のチュートリアル

AWS CLI および .NET アプリケーション

このセクションでは、 を使用して一時的な SSO トークンを生成する方法と AWS CLI、そのトークンをアプリケーションで使用する方法について説明します。このプロセスの詳細なチュートリアルについては、AWS CLI および .NET アプリケーションを使用した SSO のチュートリアルを参照してください。

を使用して SSO トークンを生成する AWS CLI

一時的な SSO トークンをプログラムで生成するだけでなく、 AWS CLI を使用してトークンを生成します。次の情報は、その方法を示しています。

前のセクションで説明したように、ユーザは SSO 対応プロファイルを作成したら、 AWS CLIからaws sso loginコマンドを実行します。SSO 対応プロファイルの名前には必ず--profileパラメータを含める必要があります。これを次の例で示します:

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

現在のトークンの有効期限が切れた後に新しい一時トークンを生成したい場合は、同じコマンドをもう一度実行できます。

生成された SSO トークンを.NET アプリケーションで使用します。

次の情報は、すでに生成されている一時トークンの使用方法を示しています。

> **Important**
>
> SSO 解決が機能するには、アプリケーションが次の NuGet パッケージを参照する必要があります。
>
> - AWSSDK.SSO
> - AWSSDK.SSOOIDC
>
> これらのパッケージを参照しないと、ランタイム例外が発生します。

アプリケーションが SSO プロファイル用のAWSCredentialsオブジェクトを作成し、 AWS CLIによって以前に生成された一時的な認証情報が読み込まれます。これはアプリケーションでの認証情報とプロファイルへのアクセスに示した方法と似ており、形式は次のとおりです。

```
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;
}
```

その後、AWSCredentialsオブジェクトはサービスクライアントのコンストラクターに渡されます。例:

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

> **Note**
>
> アプリケーションが SSO 用の[default]プロファイルを使用するように構築されている場合は、AWSCredentialsを使用して一時的な認証情報を読み込む必要はありません。この場合、アプリケーションはvar client = new AmazonS3Client();「」のようなパラメータなしで AWS サービスクライアントを作成できます。

AWS CLI および .NET アプリケーションを使用した SSO のチュートリアル

追加リソース

その他のヘルプについては、以下の資料を参照してください。

- IAM Identity Centerとは何ですか?
- IAM Identity Center を使用する AWS CLI ように を設定する
- での IAM Identity Center 認証情報の使用 AWS Toolkit for Visual Studio

チュートリアル

トピック

- .NET アプリケーションのみを使用する SSO のチュートリアル
- AWS CLI および .NET アプリケーションを使用した SSO のチュートリアル

.NET アプリケーションのみを使用する SSO のチュートリアル

このチュートリアルでは、基本アプリケーションとテスト SSO ユーザーの SSO を有効にする方法を示します。AWS CLI を使用する代わりにプログラムで一時的な SSO トークンを生成するようにアプリケーションを設定します。

このチュートリアルでは、 の SSO 機能のごく一部について説明します AWS SDK for .NET。での IAM Identity Center の使用の詳細については AWS SDK for .NET、 トピックの背景情報を参照してください。そのトピックでは、特に .NETアプリケーションのみ というサブセクションにあるこのシナリオの概要説明を参照してください。

> **Note**
>
> このチュートリアルのいくつかのステップは、 AWS Organizations や IAM Identity Center などのサービスを設定するのに役立ちます。その設定をすでに実行している場合や、コードのみに関心がある場合は、サンプルコードのセクションまでスキップできます。

前提条件

- まだの場合は、開発環境を設定します。これについては、用のツールチェーンのインストールと設定 AWS SDK for .NET や 入門 などのセクションで説明しています。
- SSO AWS アカウント のテストに使用できる 1 つ以上の を特定または作成します。このチュートリアルでは、これをテスト AWS アカウント、または単にテストアカウントと呼びます。
- SSO をテストしてくれる SSO ユーザーを指定します。これは SSO と、作成した基本的なアプリケーションを使用するユーザーです。このチュートリアルでは、あなた (開発者) でも他のユーザーでもかまいません。また、SSO ユーザーが開発環境外のコンピューターで作業するような設定もお勧めします。ただし、これは厳密には必須ではありません。
- SSO ユーザーのコンピューターには、開発環境の設定に使用したものと互換性のある.NET Framework がインストールされている必要があります。

セットアップ AWS

このセクションでは、このチュートリアルのさまざまな AWS サービスをセットアップする方法について説明します。

この設定を実行するには、まず管理者 AWS アカウント としてテストにサインインします。次に、以下の操作を実行します。

Amazon S3

Amazon S3 コンソール に移動し、無害なバケットをいくつか追加します。このチュートリアルの後半では、SSO ユーザーがこれらのバケットのリストを取得します。

AWS IAM

IAM コンソールに移動し、IAM ユーザーを数人追加します。IAM ユーザーにアクセス権限を付与する場合は、その権限をいくつかの無害な読み取り専用権限に制限してください。このチュートリアルの後半では、SSO ユーザーがこれらの IAM ユーザーのリストを取得します。

AWS Organizations

AWS Organizations コンソール に移動し、組織を有効にします。詳細については、「AWS Organizations ユーザーガイド」で「組織を作成する」を参照してください。

このアクションは、テスト AWS アカウント を管理アカウントとして組織に追加します。テストアカウントが他にもある場合は、そのアカウントを組織に招待できますが、このチュートリアルでは必須ではありません。

IAM アイデンティティセンター

IAM アイデンティティセンターコンソール に移動してSSO を有効にします。必要に応じて E メール認証を行います。詳細については、「IAM アイデンティティセンターユーザーガイド」の「IAM アイデンティティセンターを有効化する」を参照してください。

次に、以下の設定を行います。

IAM アイデンティティセンター を設定する

1. [設定] ページに移動します。「アクセスポータル URL」 を探し、その値を後で使用できるように `sso_start_url` 設定に記録します。

2. のバナーで AWS マネジメントコンソール、SSO を有効にしたときに AWS リージョン 設定された を探します。これは AWS アカウント ID の左側にあるドロップダウンメニューです。後で使用できるようにリージョンコードを `sso_region` 設定に記録しておきます。このコードは `us-east-1` のようになります。
3. 次のように SSO ユーザーを作成します。
   a. [ユーザー]ページに移動します。
   b. [ユーザーを追加] を選択し、ユーザーの ユーザー名、 メールアドレス、 名、 姓 を入力します。その後、[Next] を選択します。
   c. グループのページで [次へ] を選択し、情報を確認して [ユーザーを追加] を選択します。
4. 次のようにグループを作成します。
   a. [グループ]ページに移動します。
   b. [グループを作成] を選択し、グループの [グループ名] と [説明] を入力します。
   c. [ユーザーをグループに追加] セクションで、先ほど作成したテスト SSO ユーザーを選択します。次に、[Create group] (グループを作成) を選択します。
5. 次のようにアクセス権限セットを作成します。
   a. [権限セット] ページに移動し、[権限セットの作成] を選択します。
   b. [権限セットの種類] で [カスタム権限セット] を選択し、[次へ] を選択します。
   c. [インラインポリシー] を開き、次のポリシーを入力します。

      JSON

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

```
}
```

d. このチュートリアルでは、 権限セット名 として SSOReadOnlyRole を入力します。必要に応じて説明を追加し、 [次へ] を選択します。

e. 情報を確認してから、[Create] (作成) を選択します。

f. 後で使用できるように権限セットの名前を sso_role_name 設定に記録します。

6. AWS アカウントページに移動し、以前に組織に追加した AWS アカウントを選択します。
7. そのページの 「概要」セクションで アカウント ID を探し、後で使用できるように sso_account_id 設定に記録しておきます。
8. [ユーザーとグループ] タブを選択し、[ユーザーまたはグループの割り当て]を選択します。
9. [ユーザーとグループの割り当て]ページで [グループ] タブを選択し、先ほど作成したグループを選択して、[次へ] を選択します。
10. 前に作成した権限セットを選択し、[次へ] を選択してから [送信] を選択します。設定には少し時間がかかります。

サンプルアプリケーションを作成する

以下のアプリケーションを作成します。それらは、 SSO ユーザーのコンピューター上で実行されます。

Amazon S3 バケットを一覧表示する

AWSSDK.S3、 AWSSDK.SecurityToken に加えて NuGet パッケージ AWSSDK.SSO、AWSSDK.SSOOIDC も含めます。

```
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;
        }
    }
}
```

IAM ユーザーのリストを取得する

AWSSDK.IdentityManagement、 AWSSDK.SecurityToken に加えて NuGet パッケージ AWSSDK.SSO、 AWSSDK.SSOOIDC も含めます。

```
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;
        }
    }
```

```
}
```

これらのアプリケーションは、Amazon S3 バケットと IAM ユーザーのリストを表示するだけでなく、SSO 対応プロファイル (このチュートリアルでは `my-sso-profile`) のユーザー ID ARN を表示します。

これらのアプリケーションは、SSOAWsCredentials オブジェクトの Options プロパティにコールバックメソッドを指定することで SSO サインインタスクを実行します。

SSO ユーザーに指示します。

SSO ユーザーに E メールを確認し、SSO の招待を受け入れるように依頼します。パスワードの設定を求められます。メッセージが SSO ユーザーの受信トレイに届くまでに数分かかる場合があります。

先ほど作成したアプリケーションを SSO ユーザーに提供します。

次に、SSO ユーザーに次の操作を行わせます。

1. 共有 AWS `config`ファイルを含むフォルダが存在しない場合は、作成します。フォルダーが存在し、`.sso` という名前のサブフォルダーがある場合は、そのサブフォルダーを削除します。

   このフォルダーの場所は通常、Windows では `%USERPROFILE%\.aws`、Linux および macOS では `~/.aws` です。
2. 必要に応じて、そのフォルダに共有 AWS `config`ファイルを作成し、次のようにプロファイルを追加します。

   ```
   [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. Amazon S3 アプリケーションを実行します。
4. 表示される Web サインインページで、サインインします。招待メッセージに記載されているユーザー名と、メッセージに応じて作成されたパスワードを使用します。
5. サインインが完了すると、アプリケーションには S3 バケットのリストが表示されます。

6. IAMアプリケーションを実行します。アプリケーションは IAM ユーザーのリストを表示します。これは、2 回目のサインインが行われなかった場合でも同様です。IAM アプリケーションは以前に作成された一時トークンを使用します。

クリーンアップ

このチュートリアルで作成したリソースを保持しない場合は、リソースをクリーンアップします。これらは、ファイル AWS やフォルダなどの開発環境のリソースである場合があります。

AWS CLI および .NET アプリケーションを使用した SSO のチュートリアル

このチュートリアルでは、基本的な .NET アプリケーションとテスト SSO ユーザーに対して SSO を有効化する方法を示します。を使用して、プログラムで AWS CLI 生成するのではなく、一時的な SSO トークンを生成します。 ???

このチュートリアルでは、 の SSO 機能のごく一部について説明します AWS SDK for .NET。での IAM Identity Center の使用の詳細については AWS SDK for .NET、 トピックの背景情報を参照してください。そのトピックでは、特に AWS CLI および .NET アプリケーション というサブセクションにあるこのシナリオの概要説明を参照してください。

> **Note**
>
> このチュートリアルのいくつかのステップは、 AWS Organizations や IAM Identity Center などのサービスを設定するのに役立ちます。これらの設定をすでに実行している場合、またはコードのみに関心がある場合は、サンプルコード のセクションまでスキップできます。

前提条件

- まだの場合は、開発環境を設定します。これについては、用のツールチェーンのインストールと設定 AWS SDK for .NET や 入門 などのセクションで説明しています。
- SSO AWS アカウント のテストに使用できる 1 つ以上の を特定または作成します。このチュートリアルでは、これをテスト AWS アカウント、または単にテストアカウントと呼びます。
- SSO をテストしてくれる SSO ユーザーを指定します。これは SSO と、作成した基本的なアプリケーションを使用するユーザーです。このチュートリアルでは、あなた (開発者) でも他のユーザーでもかまいません。また、SSO ユーザーが開発環境外のコンピューターで作業するような設定もお勧めします。ただし、これは厳密には必須ではありません。

- SSO ユーザーのコンピューターには、開発環境の設定に使用したものと互換性のある.NET Framework がインストールされている必要があります。

- AWS CLI バージョン 2 が SSO ユーザーのコンピュータにインストールされていることを確認します。これは、コマンドプロントまたはターミナルで `aws --version` を実行して確認できます。

セットアップ AWS

このセクションでは、このチュートリアルのさまざまな AWS サービスをセットアップする方法について説明します。

この設定を実行するには、まず管理者 AWS アカウント としてテストにサインインします。次に、以下の操作を実行します。

Amazon S3

Amazon S3 コンソール に移動し、無害なバケットをいくつか追加します。このチュートリアルの後半では、SSO ユーザーがこれらのバケットのリストを取得します。

AWS IAM

IAM コンソールに移動し、IAM ユーザーを数人追加します。IAM ユーザーにアクセス権限を付与する場合は、その権限をいくつかの無害な読み取り専用権限に制限してください。このチュートリアルの後半では、SSO ユーザーがこれらの IAM ユーザーのリストを取得します。

AWS Organizations

AWS Organizations コンソール に移動し、組織を有効にします。詳細については、「AWS Organizations ユーザーガイド」で「組織を作成する」を参照してください。

このアクションは、テスト AWS アカウント を管理アカウントとして組織に追加します。テストアカウントが他にもある場合は、そのアカウントを組織に招待できますが、このチュートリアルでは必須ではありません。

IAM アイデンティティセンター

IAM アイデンティティセンターコンソール に移動してSSO を有効にします。必要に応じて E メール認証を行います。詳細については、「IAM アイデンティティセンターユーザーガイド」の「IAM アイデンティティセンターを有効化する」を参照してください。

次に、以下の設定を行います。

IAM アイデンティティセンター を設定する

1. [設定] ページに移動します。「アクセスポータル URL」 を探し、その値を後で使用できるように `sso_start_url` 設定に記録します。
2. のバナーで AWS マネジメントコンソール、SSO を有効にしたときに AWS リージョン 設定された を探します。これは AWS アカウント ID の左側にあるドロップダウンメニューです。後で使用できるようにリージョンコードを `sso_region` 設定に記録しておきます。このコードは `us-east-1` のようになります。
3. 次のように SSO ユーザーを作成します。
   a. [ユーザー]ページに移動します。
   b. [ユーザーを追加] を選択し、ユーザーの ユーザー名、 メールアドレス、 名、 姓 を入力します。その後、[Next] を選択します。
   c. グループのページで [次へ] を選択し、情報を確認して [ユーザーを追加] を選択します。
4. 次のようにグループを作成します。
   a. [グループ]ページに移動します。
   b. [グループを作成] を選択し、グループの [グループ名] と [説明] を入力します。
   c. [ユーザーをグループに追加] セクションで、先ほど作成したテスト SSO ユーザーを選択します。次に、[Create group] (グループを作成) を選択します。
5. 次のようにアクセス権限セットを作成します。
   a. [権限セット] ページに移動し、[権限セットの作成] を選択します。
   b. [権限セットの種類] で [カスタム権限セット] を選択し、[次へ] を選択します。
   c. [インラインポリシー] を開き、次のポリシーを入力します。

      JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
```

```
                "s3:ListAllMyBuckets",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

d. このチュートリアルでは、 権限セット名 として SSOReadOnlyRole を入力します。必要に応じて説明を追加し、 [次へ] を選択します。

e. 情報を確認してから、[Create] (作成) を選択します。

f. 後で使用できるように権限セットの名前を sso_role_name 設定に記録します。

6. AWS アカウントページに移動し、先に組織に追加した AWS アカウントを選択します。
7. そのページの 「概要」セクションで アカウント ID を探し、後で使用できるように sso_account_id 設定に記録しておきます。
8. [ユーザーとグループ] タブを選択し、[ユーザーまたはグループの割り当て]を選択します。
9. [ユーザーとグループの割り当て]ページで [グループ] タブを選択し、先ほど作成したグループを選択して、[次へ] を選択します。
10. 前に作成した権限セットを選択し、[次へ] を選択してから [送信] を選択します。設定には少し時間がかかります。

サンプルアプリケーションを作成する

以下のアプリケーションを作成します。それらは、 SSO ユーザーのコンピューター上で実行されます。

Amazon S3 バケットを一覧表示する

AWSSDK.S3、 AWSSDK.SecurityToken に加えて NuGet パッケージ AWSSDK.SSO、AWSSDK.SSOOIDC も含めます。

```
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;
        }
    }
}
```

IAM ユーザーのリストを取得する

AWSSDK.IdentityManagement、 AWSSDK.SecurityToken に加えて NuGet パッケージ AWSSDK.SSO、 AWSSDK.SSOOIDC も含めます。

```
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;
        }
    }
}
```

これらのアプリケーションは、Amazon S3 バケットと IAM ユーザーのリストを表示するだけでなく、SSO 対応プロファイル (このチュートリアルでは `my-sso-profile`) のユーザー ID ARN を表示します。

SSO ユーザーに指示します。

SSO ユーザーに E メールを確認し、SSO の招待を受け入れるように依頼します。パスワードの設定を求められます。メッセージが SSO ユーザーの受信トレイに届くまでに数分かかる場合があります。

先ほど作成したアプリケーションを SSO ユーザーに提供します。

次に、SSO ユーザーに次の操作を行わせます。

1. 共有 AWS `config`ファイルを含むフォルダが存在しない場合は、作成します。フォルダーが存在し、`.sso` という名前のサブフォルダーがある場合は、そのサブフォルダーを削除します。

   このフォルダーの場所は通常、Windows では `%USERPROFILE%\.aws`、Linux および macOS では `~/.aws` です。
2. 必要に応じて、そのフォルダに共有 AWS `config`ファイルを作成し、次のようにプロファイルを追加します。

   ```
   [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. Amazon S3 アプリケーションを実行します。ランタイム例外が表示されます。
4. 次の AWS CLI コマンドを実行します。

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

5. 表示される Web サインインページで、サインインします。招待メッセージに記載されているユーザー名と、メッセージに応じて作成されたパスワードを使用します。
6. Amazon S3 アプリケーションを再実行します。これで、アプリケーションに S3 バケットのリストが表示されます。
7. IAMアプリケーションを実行します。アプリケーションは IAM ユーザーのリストを表示します。これは、2 回目のサインインが行われなかった場合でも同様です。IAM アプリケーションは以前に作成された一時トークンを使用します。

クリーンアップ

このチュートリアルで作成したリソースを保持しない場合は、リソースをクリーンアップします。これらは、ファイル AWS やフォルダなどの開発環境のリソースである場合があります。

アプリケーションを にデプロイする AWS

開発マシン上でクラウド ネイティブ .NET Core アプリケーションまたはサービスを開発した後、それを AWSにデプロイする必要があります。これを行うには、 AWS マネジメントコンソール または CloudFormation や などの特定のサービスを使用します AWS Cloud Development Kit (AWS CDK)。デプロイの目的で作成された AWS ツールを使用することもできます。これらのツールを使用すると、次のことを実行できます。

.NET CLI からデプロイする

.NET CLI には、次の AWS ツールを使用してアプリケーションをデプロイできます AWS。

- AWS .NET CLI 用のデプロイツール - AWS App Runner、Amazon Elastic Container Service (Amazon ECS)、および へのデプロイをサポートしますAWS Elastic Beanstalk。
- AWS Lambda Tools for .NET CLI - AWS Lambda プロジェクトのデプロイをサポートします。

IDE ツールキットからデプロイする

AWS ツールキットを使用して、選択した IDE から直接アプリケーションをデプロイできます。

- AWS Toolkit for Visual Studio

  > **Note**
  >
  > ツールキットの「Publish to AWS」機能は、.NET CLI 用の AWS デプロイツールと同じ機能を公開します。詳細については、AWS Toolkit for Visual Studio DD ユーザー ガイドの「AWSに公開」を参照してください。

- AWS Toolkit for JetBrains

  AWS 「サーバーレスアプリケーションの使用」および「 の使用 AWS App Runner」を参照してください。

- AWS Toolkit for VS Code

  「サーバーレス アプリケーションの操作」および「AWS App Runner BB の使用」を参照してください。

- AWS Toolkit for Azure DevOps

ユースケース

以下のセクションでは、.NET CLI を使用してアプリケーションをデプロイする方法など、特定のタイプのアプリケーションのユースケースシナリオについて説明します。

- ASP.NET Core Apps
- .NET コンソールアプリ
- Blazor WebAssembly アプリ
- AWS Lambda プロジェクト

ASP.NET Core Apps

.NET CLI AWS 展開ツール は、ASP.NET アプリケーションの展開に役立ち、展開プロセスをガイドします。.NET CLI 用のインタラクティブなツールで、最小限の AWS の知識で.NET アプリケーションをデプロイできます。

デプロイツールには、次の機能があります。

- アプリケーションのコンピューティングレコメンデーション - コンピューティングレコメンデーションを取得し、アプリケーションに最適な AWS コンピューティングについて説明します。
- Dockerfile 生成 - このツールは、必要に応じて Dockerfile を生成するか、既存の Dockerfile を使用します。
- 自動パッケージングとデプロイ – このツールはデプロイアーティファクトを構築し、生成された AWS CDK デプロイプロジェクトを使用してインフラストラクチャをプロビジョニングし、選択した AWS コンピューティングにアプリケーションをデプロイします。
- 反復可能で共有可能なデプロイ — 特定のユースケースに合わせて AWS CDK デプロイプロジェクトを生成および変更できます。プロジェクトをバージョン管理し、チームと共有して反復可能なデプロイメントを行うこともできます。
- .NET AWS CDK の学習に役立つ - このツールは、 など、基盤となる AWS ツールを徐々に学習するのに役立ちます AWS CDK。

AWS Deploy Tool では、ASP.NET Core アプリケーションを次の AWS サービスにデプロイできます。

- を使用した Amazon ECS Service AWS Fargate - AWS Fargate サーバーレスコンピューティングエンジンによって管理されるコンピューティング能力を使用して、Amazon Elastic Container Service (Amazon ECS) へのウェブアプリケーションのデプロイをサポートします。
- AWS App Runner - フルマネージド型サービスへのデプロイをサポートします。これにより、コンテナ化されたウェブアプリケーションや API を大規模にデプロイできます。インフラストラクチャの経験は必要ありません。
- AWS Elastic Beanstalk - デベロッパーが Web アプリケーションと API をフルマネージド環境に大規模に展開することが容易なサービスへの展開をサポートします。インフラストラクチャの経験は必要ありません。

詳細については、ツールの概要 をご覧ください。そこから始めるには、[ドキュメンテーション]、[はじめに] に移動し、[インストール方法] を選択してインストール手順を確認してください。

.NET コンソールアプリ

.NET CLI 用の AWS Deploy Tool では、.NET コンソールアプリケーションをサービスとして、またはスケジュールされたタスクをコンテナイメージとして Linux にデプロイする方法を確認できます。アプリケーションに Dockerfile がない場合は、ツールが自動的に生成します。それ以外の場合は、既存の Dockerfile が使用されます。

デプロイツールには、次の機能があります。

- アプリケーションのコンピューティングレコメンデーション - コンピューティングレコメンデーションを取得し、アプリケーションに最適な AWS コンピューティングについて説明します。
- Dockerfile 生成 - このツールは、必要に応じて Dockerfile を生成するか、既存の Dockerfile を使用します。
- 自動パッケージングとデプロイ – このツールはデプロイアーティファクトを構築し、生成された AWS CDK デプロイプロジェクトを使用してインフラストラクチャをプロビジョニングし、選択した AWS コンピューティングにアプリケーションをデプロイします。
- 反復可能で共有可能なデプロイ – 特定のユースケースに合わせて AWS CDK デプロイプロジェクトを生成および変更できます。プロジェクトをバージョン管理し、チームと共有して反復可能なデプロイメントを行うこともできます。
- .NET AWS CDK の学習に役立つ - このツールは、 など、基盤となる AWS ツールを徐々に学習するのに役立ちます AWS CDK。

AWS デプロイツールは、次の AWS サービスへの .NET コンソールアプリケーションのデプロイをサポートします。

- を使用した Amazon ECS Service AWS Fargate - AWS Fargate サーバーレスコンピューティングエンジンによって管理されるコンピューティング能力を使用して、Amazon Elastic Container Service (Amazon ECS) への .NET アプリケーション as a Service (バックグラウンドプロセッサなど) のデプロイをサポートします。
- を使用した Amazon ECS スケジュールされたタスク AWS Fargate - AWS Fargate サーバーレスコンピューティングエンジンによって管理されるコンピューティング能力を使用して、スケジュールされたタスク (end-of-dayのプロセスなど) としての .NET アプリケーションの Amazon ECS へのデプロイをサポートします。

詳細については、ツールの概要 をご覧ください。そこから始めるには、[ドキュメンテーション]、[はじめに] に移動し、[インストール方法] を選択してインストール手順を確認してください。

Blazor WebAssembly アプリ

.NET CLI 用のAWS デプロイツール を使用すると、Amazon CloudFront を使用してコンテンツネットワーク配信に Blazor WebAssembly アプリケーションを Amazon S3 でホストできます。アプリケーションはウェブホスティング用の S3 バケットにデプロイされます。このツールは S3 バケットを作成して設定し、Blazor アプリケーションをそのバケットにアップロードします。

デプロイツールには、次の機能があります。

- 自動パッケージングとデプロイ – このツールはデプロイアーティファクトを構築し、生成された AWS CDK デプロイプロジェクトを使用してインフラストラクチャをプロビジョニングし、選択した AWS コンピューティングにアプリケーションをデプロイします。
- 反復可能で共有可能なデプロイ – 特定のユースケースに合わせて AWS CDK デプロイプロジェクトを生成および変更できます。プロジェクトをバージョン管理し、チームと共有して反復可能なデプロイメントを行うこともできます。
- .NET AWS CDK の学習に役立つ - このツールは、 など、基盤となる AWS ツールを徐々に学習するのに役立ちます AWS CDK。

詳細については、ツールの概要 をご覧ください。そこから始めるには、[ドキュメンテーション]、[はじめに] に移動し、[インストール方法] を選択してインストール手順を確認してください。

AWS Lambda プロジェクト

AWS Lambda は、サーバーのプロビジョニングや管理を行わずにコードを実行できるようにするコンピューティングサービスです。可用性の高いコンピューティングインフラストラクチャでコードを実行し、コンピューティングリソースの管理をすべて担当します。Lambda の詳細については、「AWS Lambda デベロッパーガイド」のAWS 「Lambda とは」を参照してください。

.NET コアコマンドラインインターフェイス (CLI) を使用して、Lambda 関数をデプロイできます。

トピック

- 前提条件
- 使用できる Lambda コマンド
- デプロイ手順

前提条件

.NET CLI を使用して Lambda 関数をデプロイする前に、次の前提条件を満たす必要があります。

- .NET CLI がインストールされていることを確認します。例: `dotnet --version`。必要に応じて https://dotnet.microsoft.com/download にアクセスしてインストールしてください。
- .NET CLI をセットアップして Lambda を操作します。その方法の説明については、「AWS Lambda デベロッパーガイド」の「.NET Lambda Global CLI を使用する」を参照してください。その手順に含まれるデプロイコマンドは次のとおりです。

  ```
  dotnet lambda deploy-function MyFunction --function-role role
  ```

  この演習で IAM ロールを作成する方法がわからない場合、`--function-role role` 部分は含めないでください。このツールを使用すると、新しいロールの作成に役立ちます。

使用できる Lambda コマンド

.NET CLI で使用できる Lambda コマンドを一覧表示するには、コマンドプロンプトまたはターミナルを開いて、`dotnet lambda --help` を入力します。そのコマンドの出力は、次のようになります。

```
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>
```

出力には、現在使用できるコマンドがすべて表示されます。

デプロイ手順

次の手順では、 AWS Lambda .NET プロジェクトを作成したことを前提としています。この手順では、プロジェクトに DotNetCoreLambdaTest という名前を付けます。

1. コマンド プロンプトまたはターミナルを開き、.NET Lambda プロジェクト ファイルが含まれるフォルダーに移動します。
2. dotnet lambda deploy-function と入力します。
3. プロンプトが表示されたら、 AWS リージョン (Lambda 関数がデプロイされるリージョン) を入力します。
4. プロンプトされたら、デプロイする関数の名前を入力します。例：DotNetCoreLambdaTest。これは、 AWS アカウント に既に存在する関数の名前、またはまだそこにデプロイされていない関数の名前である可能性があります。
5. プロンプトがされたら、関数が実行されるときに Lambda が継承する IAM ロールを選択または作成します。

正常に完了すると、[New Lambda function created (新しい Lambda 関数が作成されました)] のメッセージが表示されます。

```
Executing publish command
...
(etc.)
New Lambda function created
```

アカウントに既に存在する関数をデプロイする場合、デプロイ機能は AWS リージョンのみを求めます (必要な場合)。この場合、コマンド出力は `Updating code for existing function` で終わります。

デプロイされた Lambda 関数は、すぐに使用できる状態になっています。詳細については、「AWS Lambda の使用例」を参照してください。

Lambda は Lambda 関数を自動的にモニタリングし、Amazon CloudWatch からメトリクスを報告します。。Lambda 関数のモニタリングおよびトラブルシューティングについては、「Lambda アプリケーションのモニタリングとトラブルシューティング」を参照してください。

からの AWS サービスの呼び出し AWS SDK for .NET

以下のセクションには、 を使用して AWS サービス AWS SDK for .NET を操作する方法を示す例、チュートリアル、タスク、ガイドが含まれています。これらの例とチュートリアルは、 AWS SDK for .NET が提供する API に基づいています。API で使用できるクラスとメソッドを確認するには、AWS SDK for .NET API リファレンスを参照してください。

を初めて使用する場合は AWS SDK for .NET、まずシンプルなアプリケーションの作成トピックを確認してください。SDK についてわかりやすく説明しています。

その他のコード例については、GitHub の AWS コード例リポジトリ と awslabs リポジトリで見つけることができます。

開始する前に、環境をセットアップし、プロジェクトを設定したことを確認してください。また、「SDK の使用」の情報を確認してください。

トピック

- のガイド付きコード例 AWS SDK for .NET
- コンピューティングサービス AWS Lambda での の使用
- の高レベルライブラリとフレームワーク AWS SDK for .NET
- 他の AWS サービスと設定のサポート

のガイド付きコード例 AWS SDK for .NET

以下のセクションにはコード例が含まれ、その例に関するガイダンスを提供します。これらは、 を使用して AWS サービス AWS SDK for .NET を操作する方法を学ぶのに役立ちます。その他のコード例については、「コードの例」を参照してください。

> **Note**
>
> の V3 に固有のコード例 AWS SDK for .NET も利用できます。それらを見つけるには、 AWS SDK for .NET 「(V3) デベロッパーガイド」の「ガイダンスを含むコード例」と「コード例」を参照してください。ただし、SDK (最新バージョン) の V3-specificコード例を使用する場合は、 の情報に従って調整する必要がある場合がありますバージョン 4 への移行。 V4

を初めて使用する場合は AWS SDK for .NET、まずシンプルなアプリケーションの作成トピックを確認してください。SDK についてわかりやすく説明しています。

開始する前に、環境をセットアップし、プロジェクトを設定したことを確認してください。また、「SDK の使用」の情報を確認してください。

トピック

- CloudFormation を使用した へのアクセス AWS SDK for .NET
- Amazon Cognito を使用したユーザー認証
- Amazon DynamoDB NoSQL データベースの使用
- Amazon EC2 の使用
- を使用した AWS Identity and Access Management (IAM) へのアクセス AWS SDK for .NET
- Amazon Simple Storage Service インターネットストレージの使用
- Amazon Simple Notification Service を使用したクラウドからの通知の送信
- Amazon SQS を使用したメッセージング

CloudFormation を使用した へのアクセス AWS SDK for .NET

はAWS CloudFormation、 AWS インフラストラクチャのデプロイを予測どおりに繰り返し作成およびプロビジョニングする AWS SDK for .NET をサポートします。

API

AWS SDK for .NET はAPIs を提供します。 CloudFormation APIs を使用すると、テンプレートやスタックなどの CloudFormation 機能を操作できます。このセクションでは、これらの API を操作する際に活用できるパターンを示すいくつかの例を紹介します。API の完全なセットを確認するには、AWS SDK for .NET API リファレンスを参照してください(「Amazon.CloudFormation」までスクロールします)。

AWS CloudFormation APIs は AWSSDK.CloudFormation パッケージによって提供されます。

前提条件

開始する前に、環境をセットアップし、プロジェクトを設定したことを確認してください。また、「SDK の使用」の情報を確認してください。

トピック

トピック

- を使用した AWS リソースの一覧表示 AWS CloudFormation

を使用した AWS リソースの一覧表示 AWS CloudFormation

この例では、 AWS SDK for .NET を使用して CloudFormation スタック内のリソースを一覧表示する方法を示します。この例では、低レベル API を使用します。アプリケーションは引数を取らず、ユーザーの認証情報にアクセスできるすべてのスタックの情報を単に収集して、それらのスタックに関する情報を表示します。

SDK リファレンス

NuGet パッケージ:

- AWSSDK.CloudFormation

プログラミング要素:

- 名前空間 Amazon.CloudFormation

  クラス AmazonCloudFormationClient
- 名前空間 Amazon.CloudFormation.Model

  クラス ICloudFormationPaginatorFactory.DescribeStacks

  クラス DescribeStackResourcesRequest

  クラス DescribeStackResourcesResponse

  クラス Stack

  クラス StackResource

  クラス Tag

```
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;
        }
    }
```

Amazon Cognito を使用したユーザー認証

> **Note**
>
> このトピックの情報は、.NET Framework および AWS SDK for .NET バージョン 3.3 以前のプロジェクトに固有のものです。

Amazon Cognito Identity を使用すると、ユーザーの一意の ID を作成し、Amazon S3 や Amazon DynamoDB などの AWS リソースへの安全なアクセスのために認証できます。Amazon Cognito ID では、公開 ID プロバイダー (Amazon、Facebook、Twitter/Digits、Google、あるいは OpenID Connect と互換性のあるプロバイダーなど) および未認証 ID がサポートされています。また Cognito では、デベロッパーが認証した ID もサポートされており、Amazon Cognito Sync を使用したユーザーデータの同期と AWS リソースへのアクセスを活用しながら、独自のバックエンド認証プロセスを通じてユーザー登録や認証ができます。

Amazon Cognito の詳細については、Amazon Cognito デベロッパーガイド を参照してください。

以下のコード例では、Amazon Cognito ID を簡単に使用する方法を示しています。認証情報プロバイダー の例では、ユーザー ID を作成して認証する方法を示しています。CognitoAuthentication 拡張

ライブラリ の例では、CognitoAuthentication 拡張ライブラリを使用して Amazon Cognito ユーザープールの認証を行う方法を示しています。

トピック

- Amazon Cognito 認証情報プロバイダー
- Amazon CognitoAuthentication 拡張ライブラリの例

Amazon Cognito 認証情報プロバイダー

> **Note**
>
> このトピックの情報は、.NET Framework および AWS SDK for .NET バージョン 3.3 以前のプロジェクトに固有のものです。

`Amazon.CognitoIdentity.CognitoAWSCredentials`AWSSDK.CognitoIdentity NuGet パッケージにある は、Amazon Cognito と AWS Security Token Service (AWS STS) を使用して AWS 呼び出しを行う認証情報を取得する認証情報オブジェクトです。

`CognitoAWSCredentials` の設定での最初のステップは「ID プール」を作成することです。(アイデンティティプールとは、お客様のアカウントに固有のユーザー ID 情報のストアです。) 情報はクライアントプラットフォーム、デバイス、およびオペレーティングシステム間で取得可能です。これにより、ユーザーが電話でアプリの使用を開始して、後でタブレットに切り替えた場合でも、保持されたアプリ情報はそのユーザーに対して引き続き利用可能になります。Amazon Cognito コンソールから新しい ID プールを作成できます。コンソールを使用している場合は、必要なその他の情報も提供します。

- アカウント番号 - 123456789012 などの、アカウントに一意な 12 桁の数字。
- 認証されていないロール ARN - 認証されていないユーザーが引き受けるロール。たとえば、このロールは、データへの読み取り専用アクセス許可を提供できます。
- 認証されたロール ARN - 認証されたユーザーが引き受けるロール。このロールは、データへのより広範なアクセス許可を提供できます。

CognitoAWSCredentials をセットアップする

次のコード例では、認証されていないユーザーとして Amazon S3 を呼び出すことを可能にする `CognitoAWSCredentials` のセットアップ方法を示しています。これにより、ユーザーを認証する

のに必要な最小量のデータだけで呼び出すことができます。ユーザーアクセス許可はロールによって制御されるため、必要に応じてアクセスを設定することもできます。

```
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();
}
```

認証されていないユーザー AWS として を使用する

次のコード例は、認証されていないユーザー AWS として の使用を開始し、Facebook を介して認証し、Facebook 認証情報を使用するように認証情報を更新する方法を示しています。この方法を使用すると、認証されたロールを通じて、認証されたユーザーにさまざまな機能を付与できるようになります。たとえば、ユーザーが匿名でコンテンツを表示することを許可するものの、1 つ以上の設定済みプロバイダーでログオンしている場合に投稿を許可する電話アプリケーションがあります。

```
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 オブジェクトは、 AWS SDK for .NETの一部である AmazonCognitoSyncClient で使用した場合、さらに多くの機能を提供します。AmazonCognitoSyncClient と CognitoAWSCredentials の両方を使用している場合は、AmazonCognitoSyncClient を使用して呼び出しを行うときに IdentityPoolId と IdentityId のプロパティを指定する必要はありません。これらのプロパティは CognitoAWSCredentials から自動的に入力されます。次のコード例では、これとともに、IdentityId の CognitoAWSCredentials が変更されるたびに通知するイベントを示します。IdentityId は、たとえば、認証されていないユーザーから認証されたユーザーに変更する際など、場合によって変更されます。

```
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 = "..."
    });
}
```

Amazon CognitoAuthentication 拡張ライブラリの例

> **Note**
>
> このトピックの情報は、.NET Framework および AWS SDK for .NET バージョン 3.3 以前のプロジェクトに固有のものです。

CognitoAuthentication 拡張ライブラリは、.NET Core および Xamarin デベロッパーによる Amazon Cognito ユーザープールの認証プロセスを簡素化するもの

で、Amazon.Extensions.CognitoAuthentication NuGet パッケージで提供されています。このライブラリは、Amazon Cognito ID プロバイダー API に基づいて構築されており、ユーザー認証 API コールを作成して送信します。

CognitoAuthentication 拡張ライブラリの使用

Amazon Cognito には、Secure Remote Password (SRP) を通じてユーザー名とパスワードを検証する標準的な認証フローのための、組み込みの `AuthFlow` 値と `ChallengeName` 値がいくつかあります。認証フローの詳細については、「Amazon Cognito ユーザープール認証フロー」を参照してください。

以下の例では、これらの `using` ステートメントが必要になります。

```
// 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;
```

基本的な認証の使用

AnonymousAWSCredentials を使用して、署名付きリクエストを必要としない AmazonCognitoIdentityProviderClient を作成します。リージョンを指定する必要はありませんが、基盤となるコードはリージョンが提供されない場合 `FallbackRegionFactory.GetRegionEndpoint()` を呼び出します。`CognitoUserPool` および `CognitoUser` オブジェクトを作成します。ユーザーパスワードを含む `StartWithSrpAuthAsync` で `InitiateSrpAuthRequest` メソッドを呼び出します。

```
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;

}
```

チャレンジを使用した認証

NewPasswordRequired や多要素認証 (MFA) などのチャレンジを使用して認証フローを継続することも、より簡単な方法です。唯一の要件は、CognitoAuthentication オブジェクト、SRP 用のユーザーパスワード、およびユーザーが入力することにより取得される、次のチャレンジに必要な情報です。次のコードは、チャレンジタイプを確認し、認証フロー中の MFA および NewPasswordRequired チャレンジに対する適切な応答を取得する方法の 1 つを示しています。

前と同じように、基本的な認証リクエストと await および AuthFlowResponse を実行します。レスポンスを受信すると、返された AuthenticationResult オブジェクトをループします。ChallengeName タイプが NEW_PASSWORD_REQUIRED の場合は、RespondToNewPasswordRequiredAsync メソッドを呼び出します。

```
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.");
    }

}
```

認証後に AWS リソースを使用する

CognitoAuthentication ライブラリを使用してユーザーが認証されたら、次のステップは、ユーザーに適切な AWS リソースへのアクセスを許可することです。そのためには、Amazon Cognito フェデレーテッド ID コンソールを通じて ID プールを作成する必要があります。プロバイダーとして作成した Amazon Cognito ユーザープールを、その poolID および clientID を使用して指定することにより、Amazon Cognito ユーザープールのユーザーがアカウントに接続された AWS リソースにアクセスすることを許可できます。異なるロールを指定して、認証されていないユーザーと認証されたユーザーの両方が異なるリソースにアクセスするようにもできます。IAM コンソールでこれらのルールを変更できます。ロールのアタッチされたポリシーの [アクション] フィールドで、アクセス許可を追加または削除できます。次に、適切な ID プール、ユーザープール、Amazon Cognito ユーザー情報を使用して、さまざまな AWS リソースを呼び出すことができます。次の例は、SRP で認証されたユーザーが、関連付けられた ID プールのロールによって許可された異なる Amazon S3 バケットにアクセスする方法を示しています。

```
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);
        }
    }
}
```

その他の認証オプション

SRP に加えて、NewPasswordRequired、MFA、CognitoAuthentication 拡張ライブラリは、以下の簡単な認証フローを提供しています。

- Custom - `StartWithCustomAuthAsync(InitiateCustomAuthRequest customRequest)` を呼び出すことで開始する
- RefreshToken - `StartWithRefreshTokenAuthAsync(InitiateRefreshTokenAuthRequest refreshTokenRequest)` を呼び出すことで開始する
- RefreshTokenSRP - `StartWithRefreshTokenAuthAsync(InitiateRefreshTokenAuthRequest refreshTokenRequest)` を呼び出すことで開始する
- AdminNoSRP - `StartWithAdminNoSrpAuthAsync(InitiateAdminNoSrpAuthRequest adminAuthRequest)` を呼び出すことで開始する

フローに応じて適切なメソッドを呼び出します。その後、各メソッド呼び出しの `AuthFlowResponse` オブジェクトで示されるように、ユーザーにチャレンジを表示し続けます。また、MFA チャレンジには `RespondToSmsMfaAuthAsync`、カスタムチャレンジには `RespondToCustomAuthAsync` など、適切なレスポンスメソッドを呼び出します。

Amazon DynamoDB NoSQL データベースの使用

> **Note**
>
> これらのトピックのプログラミングモデルは.NET Framework と.NET (Core) の両方に存在しますが、呼び出し規則は同期か非同期かで異なります。

は、 が提供する高速 NoSQL データベースサービスである Amazon DynamoDB AWS SDK for .NET をサポートしています AWS。SDK では、DynamoDB との通信用に 3 つのプログラムモデルが提供されています。低レベルモデル、ドキュメントモデル、オブジェクト永続性モデルです。

以下では、これらのモデルとその API について紹介し、それぞれの使用方法と用途の例を示します。また、 AWS SDK for .NETでの追加の DynamoDB プログラミングリソースへのリンクも提供します。

低レベルモデル

低レベルプログラミングモデルは、DynamoDB サービスの直接呼び出しをラップします。このモデルには、Amazon.DynamoDBv2 名前空間からアクセスします。

低レベルモデルは、3 つのモデルの中で最も多くのコードを記述する必要があります。例えば、.NET データ型を DynamoDB の同等の型に変換する必要があります。ただし、このモデルを使用するとほとんどの機能にアクセスできます。

以下の例では、低レベルモデルを使用して DynamoDB でのテーブルの作成、テーブルの変更、テーブルへの項目の挿入を行う方法を示します。

テーブルの作成

次の例では、CreateTable クラスの AmazonDynamoDBClient メソッドを使用してテーブルを作成します。CreateTable メソッドでは、必要な項目属性名、プライマリキーの定義、スループット容量などの特性を含む CreateTableRequest クラスのインスタンスを使用します。CreateTable メソッドは、CreateTableResponse クラスのインスタンスを返します。

```
// 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);
}
```

テーブルを変更する準備が整っていることの確認

テーブルを変更または修正する前に、テーブルを変更する準備が整っていることを確認する必要があります。次の例では、低レベルモデルを使用して DynamoDB のテーブルの準備が整っていることを確認する方法を示します。この例では、チェック対象のテーブルは `DescribeTable` クラスの `AmazonDynamoDBClient` メソッドを使用して参照されています。5 秒ごとに、コードはテーブルの `TableStatus` プロパティの値を調べます。ステータスが `ACTIVE` に設定されると、テーブルは変更できる状態です。

```
// 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);
```

テーブルへの項目の挿入

次の例では、低レベルモデルを使用して DynamoDB のテーブルに 2 つの項目を挿入します。各項目は、PutItem クラスのインスタンスを使用して、AmazonDynamoDBClient クラスの PutItemRequest メソッドによって挿入されます。PutItemRequest クラスの 2 つのインスタンスはそれぞれ、項目を挿入するテーブルの名前と一連の項目属性値を取得します。

```
// 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);
```

ドキュメントモデル

ドキュメントプログラミングモデルは、DynamoDB のデータを操作する簡単な手段を提供します。このモデルは、特にテーブルおよびテーブル内の項目にアクセスすることを目的に作られています。このモデルには、Amazon.DynamoDBv2.DocumentModel 名前空間からアクセスします。

低レベルプログラミングモデルと比べると、ドキュメントモデルの方が DynamoDB データに対するコーディングが容易です。例えば、多くの .NET データ型を DynamoDB の同等のデータ型に変換する必要はありません。ただし、このモデルでは、低レベルプログラミングモデルほど多くの機能にはアクセスできません。例えば、このモデルを使用して、テーブルの項目を作成、取得、更新、削除することはできます。しかし、テーブルを作成するには、低レベルモデルを使用する必要があります。オブジェクト永続性モデルと比較すると、このモデルの方が .NET オブジェクトを保存、ロード、クエリするために多くのコードを作成する必要があります。

DynamoDB ドキュメントプログラミングモデルの詳細については、Amazon DynamoDB デベロッパーガイドの「.NET: ドキュメントモデル」を参照してください。

以下のセクションでは、目的の DynamoDB テーブルの表現を作成する方法と、ドキュメントモデルを使用してテーブルに項目を挿入し、テーブルから項目を取得する方法の例を示します。

表の表現の作成

ドキュメントモデルを使用してデータオペレーションを実行するには、最初に メソッドを呼び出して、特定のテーブルを表す Tableクラスのインスタンスを作成します。これには、主に 2 つの方法があります。

LoadTable (メソッド)

1 つ目のメカニズムは、次の例のように、Tableクラスの静的LoadTableメソッドの 1 つを使用することです。

```
var client = new AmazonDynamoDBClient();
Table table = Table.LoadTable(client, "Reply");
```

> **Note**
>
> このメカニズムは機能しますが、特定の条件下では、コールドスタートやスレッドプールの動作が原因で、レイテンシーが増えたり、デッドロックが発生したりすることがあります。これらの動作の詳細については、ブログ記事「AWS SDK for .NETのDynamoDB 初期化パターンの改善」を参照してください。

テーブルビルダー

代替メカニズムであるTableBuilderクラスは、AWSSDK.DynamoDBV2 NuGet パッケージのバージョン 3.7.203 で導入されました。このメカニズムでは、特定の暗黙的なメソッド呼び出し (具体的にはDescribeTableメソッド) を削除することで、上記の動作に対処できます。このメカニズムは、次の例のような方法で使用されます。

```
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();
```

この代替メカニズムの詳細については、ブログ記事「 AWS SDK for .NETの DynamoDB 初期化パターンの改善」をもう一度参照してください。

テーブルへの項目の挿入

次の例では、PutItemAsync クラスの Table メソッドを使用してテーブルにリプライを挿入しています。PutItemAsync メソッドは、Document クラスのインスタンスを受け取ります。Document クラスは、初期化された属性の単純なコレクションです。

```
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);
```

テーブルからの項目の取得

次の例では、GetItemAsync クラスの Table メソッドを使用してリプライを取得しています。取得するリプライを特定するために、GetItemAsync メソッドでは対象リプライのハッシュおよび範囲プライマリキーを使用しています。

```
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"]);
```

前述の例では、テーブル値を WriteLine メソッドの文字列に暗黙的に変換しています。DynamoDBEntryクラスのさまざまな 「As[type]」 メソッドを使用すると、明示的に変換できます。たとえば、AsGuid() メソッドを使用して、Id の値を Primitive データ型から GUID に明示的に変換できます。

```
var guid = reply["Id"].AsGuid();
```

オブジェクト永続性モデル

オブジェクト永続性プログラミングは、特に DynamoDB での .NET オブジェクトの保存、ロード、クエリを目的として作られています。このモデルには、Amazon.DynamoDBv2.DataModel 名前空間からアクセスします。

3 つのモデルの中で、オブジェクト永続性モデルは、DynamoDB データの保存、ロード、クエリに関してはコーディングが最も簡単です。例えば、DynamoDB のデータ型を直接操作できます。ただし、このモデルでアクセスできるのは、DynamoDB に .NET オブジェクトを保存、ロード、クエリする操作だけです。例えば、このモデルを使用して、テーブルの項目を作成、取得、更新、削除することはできます。ただし、最初に低レベルモデルを使用してテーブルを作成してから、このモデルを使用して .NET クラスをテーブルにマッピングする必要があります。

DynamoDB オブジェクト永続性プログラミングモデルの詳細については、「Amazon DynamoDB デベロッパーガイド」の「.NET: オブジェクト永続性モデル」を参照してください。

次の例は、DynamoDB 項目を表す .NET クラスを定義する方法、.NET クラスのインスタンスを使用して項目を DynamoDB テーブルに挿入する方法、.NET クラスのインスタンスを使用してテーブルから項目を取得する方法を示しています。

テーブルで項目を表す .NET クラスの定義

次のクラス定義の例では、DynamoDBTable 属性はテーブル名を指定し、DynamoDBHashKey 属性と DynamoDBRangeKey 属性はテーブルのハッシュと範囲のプライマリキーをモデル化します。DynamoDBGlobalSecondaryIndexHashKey属性は、特定の作成者による返信のクエリを作成できるように定義されています。

```
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; }
}
```

オブジェクト永続性モデルのコンテキストを作成します。

DynamoDB のオブジェクト永続プログラミングモデルを使用するには、コンテキストを作成する必要があります。このクラスから DynamoDB に接続して、テーブルにアクセスし、各種のオペレーションとクエリを実行することができます。

基本コンテキスト

次の例は、最も基本的なコンテキストを作成する方法を示しています。

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;

var client = new AmazonDynamoDBClient();
var context = new DynamoDBContext(client);
```

DisableFetchingTableMetadata プロパティによるコンテキスト

次の例は、DescribeTableメソッドが暗黙的に呼び出されないように、DynamoDBContextConfigクラスのDisableFetchingTableMetadataプロパティを追加で設定する方法を示しています。

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
```

```
var client = new AmazonDynamoDBClient();
var context = new DynamoDBContext(client, new DynamoDBContextConfig
{
    DisableFetchingTableMetadata = true
});
```

最初の例のようにDisableFetchingTableMetadataプロパティが false (デフォルト) に設定されている場合は、テーブル項目のキーとインデックスの構造を記述する属性をReplyクラスから省略できます。代わりに、これらの属性はDescribeTableメソッドを暗黙的に呼び出すことで推測されます。2 番目の例に示すように、 DisableFetchingTableMetadata が true に設定されている場合、SaveAsync や QueryAsync などのオブジェクト永続モデルのメソッドは、Reply クラスで定義された属性に完全に依存します。この場合、DescribeTableメソッドの呼び出しは行われません。

> **Note**
>
> 特定の条件下では、コールドスタートやスレッドプールの動作が原因で、DescribeTableメソッドを呼び出すとレイテンシーが増えたり、デッドロックが発生したりすることがあります。このため、そのメソッドの呼び出しは避けたほうが有利な場合があります。
> これらの動作の詳細については、ブログ記事「 AWS SDK for .NETのDynamoDB 初期化パターンの改善」を参照してください。

.NET クラスのインスタンスの使用によるテーブルへの項目の挿入

上の例では、項目は DynamoDBContext クラスの SaveAsync メソッドによって挿入されます。このメソッドは、項目を表す .NET クラスの初期化されたインスタンスを受け取ります

```
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"
});
```

.NET クラスのインスタンスを使用してテーブルから項目を取得する

この例では、DynamoDBContextクラスのQueryAsyncメソッドを使用して「Author1」のすべてのレコードを検索するクエリを作成します。次に、クエリのGetNextSetAsyncメソッドを使用して項目が取得されます。

```
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);
}
```

オブジェクト永続モデルに関する追加情報

上記の例と説明には、DisableFetchingTableMetadataというDynamoDBContextクラスのプロパティが含まれている場合があります。[awsSDK.DynamoDBV2 NuGet パッケージのバージョン]

3.7.203 で導入されたこのプロパティを使用すると、コールドスタートやスレッドプールの動作が原因で、追加のレイテンシーやデッドロックが発生する可能性のある特定の条件を回避できます。詳細については、ブログ記事「 AWS SDK for .NETのDynamoDB 初期化パターンの改善」を参照してください。

このプロパティに関する追加情報は次のとおりです。

- .NET Framework を使用している場合は、このプロパティをapp.configまたはweb.configファイルにグローバルに設定できます。
- このプロパティは、次の例に示すように、AWSConfigsDynamoDBクラスを使用してグローバルに設定できます。

```
// Set the DisableFetchingTableMetadata property globally
// before constructing any context objects.
AWSConfigsDynamoDB.Context.DisableFetchingTableMetadata = true;

var client = new AmazonDynamoDBClient();
var context = new DynamoDBContext(client);
```

- クラスが依存関係で定義されている場合など、DynamoDB 属性を.NET クラスに追加できない場合があります。このような場合でも、DisableFetchingTableMetadataプロパティを利用することは可能です。そのためには、DisableFetchingTableMetadataプロパティに加えてTableBuilderクラスも使用します。TableBuilderクラスは、AWSSDK.DynamoDBV2 NuGet パッケージのバージョン 3.7.203 でも導入されました。

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

詳細情報

AWS SDK for .NET を使用して DynamoDB をプログラムする、情報と例

- DynamoDB API
- DynamoDB Series Kickoff
- DynamoDB Series - Document Model
- DynamoDB Series - Conversion Schemas
- DynamoDB Series - Object Persistence Model
- DynamoDB Series - Expressions
- Amazon DynamoDB と での式の使用 AWS SDK for .NET
- Amazon DynamoDB での JSON のサポート

低レベルモデル、情報および例

- AWS SDK for .NET 低レベル API を使用したテーブルの操作
- AWS SDK for .NET 低レベル API を使用したアイテムの操作
- AWS SDK for .NET 低レベル API を使用したテーブルのクエリ
- AWS SDK for .NET 低レベル API を使用したテーブルのスキャン
- AWS SDK for .NET 低レベル API を使用したローカルセカンダリインデックスの操作
- AWS SDK for .NET 低レベル API を使用したグローバルセカンダリインデックスの使用

ドキュメントモデル、情報および例

- DynamoDB データ型
- DynamoDBEntry

- .NET ドキュメントモデル

オブジェクト永続性モデル、情報および例

- .NET: オブジェクト永続性モデル

その他の有用な情報

- .NET Aspire を通じた Amazon DynamoDB Local での開発については、「.NET Aspire AWS との統合」を参照してください。

トピック

- Amazon DynamoDB と での式の使用 AWS SDK for .NET
- Amazon DynamoDB での JSON のサポート

Amazon DynamoDB と での式の使用 AWS SDK for .NET

> **Note**
>
> このトピックの情報は、.NET Framework および AWS SDK for .NET バージョン 3.3 以前のプロジェクトに固有のものです。

次のコード例は、 を使用して式で DynamoDB AWS SDK for .NET をプログラムする方法を示しています。式は、DynamoDB テーブルの項目から読み取る属性を示します。また、項目を書き込むときも式を使用して、満たす必要がある条件 (条件付き更新とも呼ばれます) と、属性を更新する方法を示します。更新の例として、属性を新しい値で置き換えたり、新しいデータをリストやマップに追加したりします。詳細については、「式を使用した項目の読み取りと書き込み」を参照してください。

トピック

- サンプルデータ
- 式と項目のプライマリキーを使用して単一の項目を取得する
- 式およびテーブルのプライマリキーを使用して複数の項目を取得する
- 式および他の項目属性を使って複数の項目を取得する
- 項目の出力

- 式を使用して項目を作成または置換する
- 式を使用して項目を更新する
- 式を使用して項目を削除する
- 詳細情報

サンプルデータ

このトピックのコード例では、ProductCatalog という名前の DynamoDB テーブルに存在する以下の 2 つの項目を例として使用します。これらの項目は、架空の自転車店のカタログの製品エントリに関する情報を示します。これらの項目は、「導入事例: ProductCatalog 項目」で提供されている例に基づきます。BOOL、L、M、N、NS、S、SS などのデータ型記述子は、JSON データ形式でのデータ型に対応します。

```
{
  "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."
        ]
      }
    }
  }
}
```

式と項目のプライマリキーを使用して単一の項目を取得する

次の例では、Amazon.DynamoDBv2.AmazonDynamoDBClient.GetItem メソッドと一連の式を使用して、Id が 205 である項目を取得して出力します。項目の属性のうち返されるものは、Id、Title、Description、Color、RelatedItems、Pictures、ProductReviews だけです。

```
// 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);
```

前の例で、ProjectionExpression プロパティは返される属性を指定しています。ExpressionAttributeNames プロパティで、プレースホルダー #pr は ProductReviews 属性を表し、プレースホルダー #ri は RelatedItems 属性を表します。PrintItem の呼び出しは、「項目の出力」で説明されているようにカスタム関数を参照します。

式およびテーブルのプライマリキーを使用して複数の項目を取得する

次の例では、Amazon.DynamoDBv2.AmazonDynamoDBClient.Query メソッドと一連の式を使用して、Id が 301 で Price の値が 150 より大きい項目を取得して出力します。返される項目の属性は、Id、Title、および ThreeStar のすべての ProductReviews 属性だけです。

```
// 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("=====");
}
```

前の例で、ProjectionExpression プロパティは返される属性を指定しています。ExpressionAttributeNames プロパティで、プレースホルダー #pr は ProductReviews 属性を表し、プレースホルダー #p は Price 属性を表します。#pr.ThreeStar は、ThreeStar 属性だけを返すように指定します。ExpressionAttributeValues プロパティは、プレースホルダー :val が値 150 を表すことを指定します。FilterExpression プロパティは、#p (Price) が :val (150) より大きくなければならないことを指定します。PrintItem の呼び出しは、「項目の出力」で説明されているようにカスタム関数を参照します。

式および他の項目属性を使って複数の項目を取得する

次の例では、Amazon.DynamoDBv2.AmazonDynamoDBClient.Scan メソッドと一連の式を使用して、ProductCategory が Bike であるすべての項目を取得して出力します。返される項目の属性は、Id、Title、および ProductReviews のすべての属性だけです。

```
// 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("=====");
}
```

前の例で、ProjectionExpression プロパティは返される属性を指定しています。ExpressionAttributeNames プロパティで、プレースホルダー #pr は ProductReviews 属性を表し、プレースホルダー #pc は ProductCategory 属性を表します。ExpressionAttributeValues プロパティは、プレースホルダー :catg が値 Bike を表すことを指定します。FilterExpression プロパティは、#pc (ProductCategory) が :catg (Bike) と等しくなければならないことを示します。PrintItem の呼び出しは、「項目の出力」で説明されているようにカスタム関数を参照します。

項目の出力

次の例では、項目の属性と値を出力する方法を示します。この例は、「式と項目のプライマリキーを使用して単一の項目を取得する」、「式およびテーブルのプライマリキーを使用して複数の項目を取得する」、「式および他の項目属性を使って複数の項目を取得する」の各方法に関する前述の例で使用されています。

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

前の例では、各属性値に含まれる複数のデータ型固有のプロパティを評価して、属性を出力する正しい形式を決定しています。このようなプロパティとしては B、BOOL、BS、L、M、N、NS、NULL、S、SS などがあり、これらは JSON データ形式に対応しています。B、N、NULL、S などのプロパティでは、対応するプロパティが null ではない場合、属性は対応する null ではないデータ型になります。BS、L、M、NS、SS などのプロパティでは、Count がゼロより大きい場合、属性は対応するゼロ値ではないデータ型になります。属性のすべてのデータ型固有プロパティが null であるか、または Count がゼロに等しい場合、属性は BOOL データ型に対応します。

式を使用して項目を作成または置換する

次の例では、Amazon.DynamoDBv2.AmazonDynamoDBClient.PutItem メソッドと一連の式を使用して、Title が 18-Bicycle 301 である項目を更新します。項目が存在しない場合、新しい項目が追加されます。

```
// 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);
```

前の例で、ExpressionAttributeNames プロパティは、プレースホルダー #title が Title 属性を表すことを指定します。ExpressionAttributeValues プロパティは、プレースホルダー :product が値 18-Bicycle 301 を表すことを指定します。ConditionExpression プロパティは、#title (Title) が :product (18-Bicycle 301) と等しくなければならないことを示します。CreateItemData の呼び出しは、次のカスタム関数を参照します。

```
// 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;
}
```

前の例では、サンプルデータを含む項目の例が呼び出し元に返されます。一連の属性と対応する値は、BOOL、L、M、N、NS、S、SS のようなデータ型を使用して構成されます。これらは JSON データ形式内のものと対応しています。

式を使用して項目を更新する

次の例では、Amazon.DynamoDBv2.AmazonDynamoDBClient.UpdateItem メソッドと一連の式を使用して、Title が 18" Girl's Bike である項目の Id を 301 に変更します。

```
// 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);
```

前の例で、ExpressionAttributeNames プロパティは、プレースホルダー #title が Title 属性を表すことを指定します。ExpressionAttributeValues プロパティは、プレースホルダー :newproduct が値 18" Girl's Bike を表すことを指定します。UpdateExpression プロパティは、#title (Title) を :newproduct (18" Girl's Bike) に変更することを指定します。

式を使用して項目を削除する

次の例では、Amazon.DynamoDBv2.AmazonDynamoDBClient.DeleteItem メソッドと一連の式を使用して、Id が 301 で Title が 18-Bicycle 301 である項目を削除します。

```
// 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);
```

前の例で、ExpressionAttributeNames プロパティは、プレースホルダー #title が Title 属性を表すことを指定します。ExpressionAttributeValues プロパティは、プレースホルダー :product が値 18-Bicycle 301 を表すことを指定します。ConditionExpression プロパティ

は、`#title ( Title )` が `:product ( 18-Bicycle 301 )` と等しくなければならないことを示します。

詳細情報

詳細な説明とコード例については、以下を参照してください。

- DynamoDB Series - Expressions
- プロジェクト式を使用した項目属性へのアクセス
- 属性の名前および値でのプレースホルダーの使用
- 条件式を使用した条件の指定
- 更新式を使用した項目および属性の変更
- AWS SDK for .NET 低レベル API を使用したアイテムの操作
- AWS SDK for .NET 低レベル API を使用したテーブルのクエリ
- AWS SDK for .NET 低レベル API を使用したテーブルのスキャン
- AWS SDK for .NET 低レベル API を使用したローカルセカンダリインデックスの操作
- AWS SDK for .NET 低レベル API を使用したグローバルセカンダリインデックスの使用

Amazon DynamoDB での JSON のサポート

> **Note**
>
> このトピックの情報は、.NET Framework および AWS SDK for .NET バージョン 3.3 以前のプロジェクトに固有のものです。

は、Amazon DynamoDB を使用する際に JSON データ AWS SDK for .NET をサポートします。そのため、DynamoDB テーブルから JSON 形式のデータを取得したり、テーブルに JSON ドキュメントを挿入したりする操作を簡単に行えます。

トピック

- DynamoDB テーブルから JSON 形式のデータを取得する
- DynamoDB テーブルに JSON 形式のデータを挿入する
- DynamoDB データ型の JSON への変換

- 詳細情報

DynamoDB テーブルから JSON 形式のデータを取得する

次の例では、DynamoDB テーブルから JSON 形式のデータを取得する方法を示します。

```
// 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
//   }
```

この例では、Document クラスの ToJson メソッドを使用してテーブルの項目を JSON 形式の文字列に変換しています。項目を取得するには、Table クラスの GetItem メソッドを使用します。取得する項目を特定するには、この例では、対象項目のハッシュおよび範囲プライマリキーを GetItem メソッドで使用しています。項目を取得するテーブルを特定するために、Table クラスの LoadTable メソッドで AmazonDynamoDBClient クラスのインスタンスと DynamoDB の対象テーブルの名前を使用しています。

DynamoDB テーブルに JSON 形式のデータを挿入する

次の例では、JSON 形式を使用して DynamoDB テーブルに項目を挿入する方法を示します。

```
// 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);
```

この例では、Document クラスの FromJson メソッドを使用して JSON 形式の文字列を項目に変換しています。項目は、PutItem クラスの Table メソッドによってテーブルに挿入されます。このメソッドは、項目を含む Document クラスのインスタンスを使用します。項目を挿入するテーブルを特定するために、Table クラスの LoadTable メソッドが呼び出され、AmazonDynamoDBClient クラスのインスタンスと DynamoDB の対象テーブルの名前を指定しています。

DynamoDB データ型の JSON への変換

Document クラスの ToJson メソッドを呼び出す場合、および結果の JSON データで FromJson メソッドを呼び出して JSON データを Document クラスのインスタンスに変換する場合、一部の DynamoDB データ型は意図したとおりには変換されません。具体的には次のとおりです。

- DynamoDB のセット (SS、NS、BS 型) は、JSON の配列に変換されます。
- DynamoDB のバイナリスカラーおよびセット (B、BS 型) は、base64 でエンコードされた JSON 文字列または文字列のリストに変換されます。

  この場合は、Document クラスの DecodeBase64Attributes メソッドを呼び出して、base64 でエンコードされた JSON データを正しいバイナリ表現に置き換える必要があります。次の例では、Document クラスのインスタンスの Picture という名前の base64 でエンコードされたバイナリスカラー項目属性を、正しいバイナリ表現で置き換えています。また、この例では、Document クラスの同じインスタンスの RelatedPictures という名前の base64 でエンコードされたバイナリセット項目属性に対しても同じことを行っています。

  ```
  item.DecodeBase64Attributes("Picture", "RelatedPictures");
  ```

詳細情報

を使用して DynamoDB で JSON をプログラミングする詳細と例については AWS SDK for .NET、以下を参照してください。

- DynamoDB JSON Support
- Amazon DynamoDB Update - JSON, Expanded Free Tier, Flexible Scaling, Larger Items

Amazon EC2 の使用

は、サイズ変更可能なコンピューティング容量を提供するウェブサービスである Amazon EC2 AWS SDK for .NET をサポートしています。このコンピューティング性能を使用して、ソフトウェアシステムの構築とホストを行います。

API

AWS SDK for .NET はAPIs を提供します。 Amazon EC2 API を通じて、セキュリティグループやキーペアなどの EC2 の機能を使用できます。API により、Amazon EC2 インスタンスを制御することもできます。このセクションでは、これらの API を操作する際に活用できるパターンを示すいくつかの例を紹介します。API の完全なセットを確認するには、AWS SDK for .NET API リファレンスを参照してください (「Amazon.EC2」までスクロールします)。

Amazon EC2 API は、AWSSDK.EC2 NuGet パッケージによって提供されます。

前提条件

開始する前に、環境をセットアップし、プロジェクトを設定したことを確認してください。また、「SDK の使用」の情報を確認してください。

の例について

このセクションの例では、Amazon EC2 クライアントの使用方法と、Amazon EC2 インスタンスを管理する方法について説明します。

EC2 スポットインスタンスのチュートリアルでは、Amazon EC2 スポットインスタンスをリクエストする方法について説明します。スポットインスタンスを使用すると、未使用の EC2 性能にオンデマンド料金よりも低価格でアクセスできます。

トピック

- Amazon EC2 でのセキュリティグループの使用
- Amazon EC2 のキーペアの使用
- Amazon EC2 のリージョンとアベイラビリティーゾーンの確認
- Amazon EC2 インスタンスの使用
- Amazon EC2 スポットインスタンスのチュートリアル

Amazon EC2 でのセキュリティグループの使用

Amazon EC2 では、セキュリティグループは 1 つ以上の EC2 インスタンスのネットワークトラフィックを制御する仮想ファイアウォールとして機能します。デフォルトでは EC2 は、インバウンドトラフィックを許可しないセキュリティグループにインスタンスを関連付けます。EC2 インスタンスが特定のトラフィックを受け付けるようにするセキュリティグループを作成できます。例えば、EC2 Windows インスタンスに接続する必要がある場合は、RDP トラフィックを許可するようにセキュリティグループを設定する必要があります。

セキュリティグループの詳細については、「Amazon EC2 ユーザーガイド」の「Amazon EC2 セキュリティグループ」を参照してください。

> **Warning**
>
> EC2-Classic は 2022 年 8 月 15 日に廃止されました。EC2-Classic は、VPC への移行をお勧めします。詳細については、ブログ記事「EC2-Classic Networking is Retiring - Here's How to Prepare」を参照してください。

API の詳細と前提条件については、親セクション (Amazon EC2 の使用) を参照してください。

トピック

- セキュリティグループの列挙
- セキュリティグループの作成
- セキュリティグループの更新

セキュリティグループの列挙

この例では、 を使用してセキュリティグループ AWS SDK for .NET を列挙する方法を示します。Amazon Virtual Private Cloud ID を指定した場合、アプリケーションは指定された VPC のセキュリティグループを列挙します。指定しなかった場合、アプリケーションは単に使用可能なすべてのセキュリティグループを一覧表示します。

以下のセクションでは、この例のスニペットを確認できます。その下には、この例のコードの全文が示されており、そのままビルドして実行できます。

トピック

- セキュリティグループの列挙

- コード全文
- その他の考慮事項

セキュリティグループの列挙

次のスニペットでは、セキュリティグループを列挙します。すべてのグループ、または特定の VPC が指定されている場合にはその VPC のグループを列挙します。

このトピックの最後で、スニペットが実際に使用されている例を確認できます。

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

```
    }
```

コード全文

このセクションでは、例に関連する参考資料とコードの全文を示します。

SDK リファレンス

NuGet パッケージ:

- AWSSDK.EC2

プログラミング要素:

- 名前空間 Amazon.EC2

  クラス AmazonEC2Client
- 名前空間 Amazon.EC2.Model

  クラス DescribeSecurityGroupsRequest

  クラス DescribeSecurityGroupsResponse

  クラス Filter

  クラス SecurityGroup

コード

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

その他の考慮事項

- VPC の場合、フィルターの名前と値のペアの Name の部分が「vpc-id」に設定されている点に注目してください。この名前は、DescribeSecurityGroupsRequest クラスの Filters プロパティの記述に由来しています。

- セキュリティグループの完全なリストを取得するには、DescribeSecurityGroupsAsync をパラメータなしで使用することもできます。

- Amazon EC2 コンソールでセキュリティグループのリストを確認することにより、結果を検証できます。

セキュリティグループの作成

この例では、 AWS SDK for .NET を使用してセキュリティグループを作成する方法を示します。既存の VPC の ID を指定して、VPC 内の EC2 用のセキュリティグループを作成できます。このよう

な ID を指定しない場合、 AWS アカウントがこれをサポートしている場合、新しいセキュリティグループは EC2-Classic 用になります。

VPC ID を指定せず、 AWS アカウントが EC2-Classic をサポートしていない場合、新しいセキュリティグループはアカウントのデフォルト VPC に属します。

> **Warning**
>
> EC2-Classic は 2022 年 8 月 15 日に廃止されました。EC2-Classic は、VPC への移行をお勧めします。詳細については、ブログ記事「EC2-Classic Networking is Retiring - Here's How to Prepare」を参照してください。

以下のセクションでは、この例のスニペットを確認できます。その下には、この例のコードの全文が示されており、そのままビルドして実行できます。

トピック

- 既存のセキュリティグループを検索する
- セキュリティグループを作成する
- コード全文

既存のセキュリティグループを検索する

次のスニペットでは、指定された名前を持つ既存のセキュリティグループを指定された VPC 内で検索します。

このトピックの最後で、スニペットが実際に使用されている例を確認できます。

```
//
// 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;
    }
```

セキュリティグループを作成する

次のスニペットでは、指定された名前のグループが指定された VPC に存在しない場合に、新しいセキュリティグループを作成します。VPC が指定されず、指定された名前のグループが 1 つ以上存在する場合、スニペットは単にグループのリストを返します。

このトピックの最後で、スニペットが実際に使用されている例を確認できます。

```
    //
    // 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;
  }
```

コード全文

このセクションでは、例に関連する参考資料とコードの全文を示します。

SDK リファレンス

NuGet パッケージ:

- AWSSDK.EC2

プログラミング要素:

- 名前空間 Amazon.EC2

  クラス AmazonEC2Client
- 名前空間 Amazon.EC2.Model

  クラス CreateSecurityGroupRequest

  クラス CreateSecurityGroupResponse

  クラス DescribeSecurityGroupsRequest

  クラス DescribeSecurityGroupsResponse

  クラス Filter

  クラス SecurityGroup

コード

```
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2CreateSecGroup
{
  // = = =
```

```
      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.");
```

```
    }
  }


  // = = =
```

```
        {
          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);
    }
  }

}
```

セキュリティグループの更新

この例では、 AWS SDK for .NET を使用してセキュリティグループにルールを追加する方法を示します。特にこの例では、特定の TCP ポートでインバウンドトラフィックを許可するルールを追加し

ます。これは、EC2 インスタンスへのリモート接続などに使用できます。アプリケーションは、既存のセキュリティグループの ID、CIDR 形式の IP アドレス (またはアドレス範囲)、およびオプションで TCP ポート番号を取得します。アプリケーションは次に、指定されたセキュリティグループにインバウンドルールを追加します。

> **Note**
>
> この例を使用するには、CIDR 形式の IP アドレス (またはアドレス範囲) が必要です。お使いのローカルコンピュータの IP アドレスを取得する方法については、このトピックの最後にある追加の考慮事項を参照してください。

以下のセクションでは、この例のスニペットを確認できます。その下には、この例のコードの全文が示されており、そのままビルドして実行できます。

トピック

- インバウンドルールの追加
- コード全文
- その他の考慮事項

インバウンドルールの追加

次のスニペットでは、セキュリティグループに特定の IP アドレス (または範囲) および TCP ポートのインバウンドルールを追加します。

このトピックの最後で、スニペットが実際に使用されている例を確認できます。

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

コード全文

このセクションでは、例に関連する参考資料とコードの全文を示します。

SDK リファレンス

NuGet パッケージ:

- AWSSDK.EC2

プログラミング要素:

- 名前空間 Amazon.EC2

  クラス AmazonEC2Client
- 名前空間 Amazon.EC2.Model

  クラス AuthorizeSecurityGroupIngressRequest

  クラス AuthorizeSecurityGroupIngressResponse

  クラス IpPermission

  クラス IpRange

コード

```
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.EC2;
using Amazon.EC2.Model;
```

```
namespace EC2AddRuleForRDP
{
  // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
 = = =
  // Class to add a rule that allows inbound traffic on TCP a port
  class Program
  {
    private const int DefaultPort = 3389;

    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;
      }

      // 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.");
    }
  }


  // = = =
```

```
    //
    // 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);
    }
  }

}
```

その他の考慮事項

- ポート番号を指定しない場合、アプリケーションはデフォルトでポート 3389 を使用します。これは Windows RDP 用のポートで、Windows が動作している EC2 インスタンスに接続できるようになります。Linux が動作している EC2 インスタンスを起動する場合は、代わりに TCP ポート 22 (SSH) を使用できます。

- 例では `IpProtocol` が「tcp」にセットされている点に注目してください。`IpProtocol` の値は、IpPermission クラスの `IpProtocol` プロパティに記述されています。

- この例を使用する際に、お使いのローカルコンピュータの IP アドレスを確認する必要があるかもしれません。アドレスを取得する方法を以下にいくつか示します。
  - EC2 インスタンスに接続するローカルコンピュータが静的パブリック IP アドレスを持っている場合は、サービスを使用してそのアドレスを取得できます。サービスの一例としては http://

checkip.amazonaws.com/ があります。インバウンドトラフィックの承認の詳細については、「Amazon EC2 ユーザーガイド」の「セキュリティグループにルールを追加する」と「さまざまなユースケースのセキュリティグループルール」を参照してください。

- ローカルコンピュータの IP アドレスを取得する別の方法として、Amazon EC2 コンソールを使用できます。

  いずれかのセキュリティグループを選択し、[Inbound rules] (インバウンドルール) タブをクリックして [Edit inbound rules] (インバウンドのルールの編集) を選択します。インバウンドルールで [Source] (送信元) 列にあるドロップダウンメニューを開き、[My IP] (マイ IP) を選択して、ローカルコンピュータの IP アドレスを CIDR 形式で表示します。必ず操作を [Cancel] (キャンセル) するようにしてください。

- Amazon EC2 コンソールでセキュリティグループのリストを確認することにより、この例の結果を検証できます。

Amazon EC2 のキーペアの使用

Amazon EC2 は公開キー暗号化を使用し、ログイン情報の暗号化と復号を行います。パブリックキー暗号ではパブリックキーを使用してデータを暗号化し、受信者はプライベートキーを使用してデータを復号します。パブリックキーとプライベートキーは、キーペアと呼ばれます。EC2 インスタンスにログインする場合、インスタンスの起動時にキーペアを指定し、接続時にキーペアのプライベートキーを指定する必要があります。

EC2 インスタンスを起動する際、キーペアを作成することもできますし、他のインスタンスの起動時に既に使用済みのキーペアを使用することもできます。Amazon EC2 キーペアについては、「Amazon EC2 ユーザーガイド」の「Amazon EC2 キーペアの操作」を参照してください。

API の詳細と前提条件については、親セクション (Amazon EC2 の使用) を参照してください。

トピック

- キーペアの作成と表示
- キーペアの削除

キーペアの作成と表示

この例では、 AWS SDK for .NET を使用してキーペアを作成する方法を示します。アプリケーションは、新しいキーペア名と PEM ファイル名 (拡張子は「.pem」) を受け取ります。そしてキーペア

を作成し、プライベートキーを PEM ファイルに書き込み、使用可能なすべてのキーペアを表示します。コマンドライン引数を指定しない場合、アプリケーションは単に使用可能なすべてのキーペアを表示します。

以下のセクションでは、この例のスニペットを確認できます。その下には、この例のコードの全文が示されており、そのままビルドして実行できます。

トピック

- キーペアを作成する
- 使用可能なキーペアの表示
- コード全文
- その他の考慮事項

キーペアを作成する

次のスニペットではキーペアが作成され、指定された PEM ファイルにプライベートキーが保存されます。

このトピックの最後で、スニペットが実際に使用されている例を確認できます。

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

使用可能なキーペアの表示

次のスニペットでは、利用可能なキーペアのリストが表示されます。

このトピックの最後で、スニペットが実際に使用されている例を確認できます。

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

コード全文

このセクションでは、例に関連する参考資料とコードの全文を示します。

SDK リファレンス

NuGet パッケージ:

- AWSSDK.EC2

プログラミング要素:

- 名前空間 Amazon.EC2

  クラス AmazonEC2Client
- 名前空間 Amazon.EC2.Model

  クラス CreateKeyPairRequest

  クラス CreateKeyPairResponse

  クラス DescribeKeyPairsResponse

  クラス KeyPairInfo

コード

```
using System;
using System.Threading.Tasks;
using System.IO;
using Amazon.EC2;
using Amazon.EC2.Model;
using System.Collections.Generic;

namespace EC2CreateKeyPair
{
  // = = =
```

```
        "\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.");
    }
 }


 // = = =
```

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

```
}
```

その他の考慮事項

- この例を実行した後、Amazon EC2 コンソールで新しいキーペアを確認できます。

- キーペアの作成時に、返されたプライベートキーを必ず保存する必要があります。後でプライベートキーを取得することはできないためです。

キーペアの削除

この例では、 AWS SDK for .NET を使用してキーペアを削除する方法を示します。アプリケーションはキーペア名を受け取ります。そのキーペアが削除された後に、使用可能なすべてのキーペアが表示されます。コマンドライン引数を指定しない場合、アプリケーションは単に使用可能なすべてのキーペアを表示します。

以下のセクションでは、この例のスニペットを確認できます。その下には、この例のコードの全文が示されており、そのままビルドして実行できます。

トピック

- キーペアの削除
- 使用可能なキーペアの表示
- コード全文

キーペアの削除

次のスニペットでは、キーペアが削除されます。

このトピックの最後で、スニペットが実際に使用されている例を確認できます。

```
    //
    // 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).");
    }
```

使用可能なキーペアの表示

次のスニペットでは、利用可能なキーペアのリストが表示されます。

このトピックの最後で、スニペットが実際に使用されている例を確認できます。

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

コード全文

このセクションでは、例に関連する参考資料とコードの全文を示します。

SDK リファレンス

NuGet パッケージ:

- AWSSDK.EC2

プログラミング要素:

- 名前空間 Amazon.EC2

  クラス AmazonEC2Client
- 名前空間 Amazon.EC2.Model

  クラス DeleteKeyPairRequest

  クラス DescribeKeyPairsResponse

  クラス KeyPairInfo

コード

```
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}");
    }
  }
}
```

Amazon EC2 のリージョンとアベイラビリティーゾーンの確認

Amazon EC2 は、世界中の複数の口ケーションでホスティングされています。これらの場所は、リージョンとアベイラビリティーゾーンで構成されています。それぞれのリージョンは地理別に区別された地域であり、アベイラビリティーゾーンと呼ばれる複数の独立した口ケーションを持っています。

リージョンとアベイラビリティーゾーンの詳細については、「Amazon EC2 ユーザーガイド」の「リージョンとゾーン」を参照してください。

この例では、 を使用して EC2 クライアントに関連するリージョンとアベイラビリティーゾーンの詳細 AWS SDK for .NET を取得する方法を示します。アプリケーションは、EC2 クライアントで使用可能なリージョンとアベイラビリティーゾーンのリストを表示します。

SDK リファレンス

NuGet パッケージ:

- AWSSDK.EC2

プログラミング要素:

- 名前空間 Amazon.EC2

  クラス AmazonEC2Client
- 名前空間 Amazon.EC2.Model

  クラス DescribeAvailabilityZonesResponse

クラス DescribeRegionsResponse

クラス AvailabilityZone

クラス Region

```
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);
    }
  }
}
```

Amazon EC2 インスタンスの使用

を使用して AWS SDK for .NET 、作成、開始、終了などのオペレーションで Amazon EC2 インスタンスを制御できます。このセクションのトピックでは、その方法についての例をいくつか示します。EC2 インスタンスの詳細については、「Amazon EC2 ユーザーガイド」の「Amazon EC2 インスタンス」を参照してください。

API の詳細と前提条件については、親セクション (Amazon EC2 の使用) を参照してください。

トピック

- Amazon EC2 インスタンスの起動
- Amazon EC2 インスタンスの終了

Amazon EC2 インスタンスの起動

この例では、 を使用して AWS SDK for .NET 、同じ Amazon マシンイメージ (AMI) から 1 つ以上の同一の設定の Amazon EC2 インスタンスを起動する方法を示します。アプリケーションは指定された複数の入力を使用して EC2 インスタンスを起動し、インスタンスが「Pending」状態でなくなるまでインスタンスを監視します。

EC2 インスタンスが実行中の場合、「(オプション) インスタンスへの接続」の説明に従ってインスタンスにリモートで接続できます。

> **Warning**
>
> EC2-Classic は 2022 年 8 月 15 日に廃止されました。EC2-Classic は、VPC への移行をお勧めします。詳細については、ブログ記事「EC2-Classic Networking is Retiring - Here's How to Prepare」を参照してください。

以下のセクションでは、この例のスニペットとその他の情報を確認できます。スニペットの下には、この例のコードの全文が示されており、そのままビルドして実行できます。

トピック

- 必要な要素を集める
- インスタンスの起動
- インスタンスのモニタリング
- コード全文
- その他の考慮事項
- (オプション) インスタンスへの接続
- クリーンアップ

必要な要素を集める

EC2 インスタンスを起動するには、いくつかの要素が必要です。

- インスタンスを起動する場所である VPC。Windows インスタンスに RDP 経由で接続する場合、VPC にはインターネットゲートウェイが添付され、そのルートテーブル内にインターネットゲートウェイへのエントリを持つ必要があるのが普通です。詳細については、Amazon VPC ユーザーガイドの「インターネットゲートウェイ」を参照してください。

- インスタンスを起動する VPC 内の既存サブネットの ID。これを簡単に検索または作成するには、Amazon VPC コンソールにサインインします。または、CreateSubnetAsyncおよびDescribeSubnetsAsyncメソッドを使用してプログラムで取得することもできます。

  > **Note**
  >
  > このパラメータを指定しなかった場合、アカウントのデフォルト VPC で新しいインスタンスが起動されます。

- インスタンスを起動する VPC に属する既存のセキュリティグループの ID。詳細については、「Amazon EC2 でのセキュリティグループの使用」を参照してください。

- 新しいインスタンスに接続する場合は、前述のセキュリティグループにポート 22 で SSH トラフィックを許可する (Linux インスタンス) か、またはポート 3389 で RDP トラフィックを許可す

る (Windows インスタンス) 適切なインバウンドルールが設定されている必要があります。設定の方法については、「セキュリティグループの更新」を参照してください (トピックの最後近くにある「その他の考慮事項」を含む)。

- インスタンスの作成に使用する Amazon マシンイメージ (AMI)。AMI の詳細については、「Amazon EC2 ユーザーガイド」の「Amazon マシンイメージ (AMI)」を参照してください。特に、「AMI の検索」と「共有 AMI」を参照してください。

- 既存の EC2 キーペア名。これは新しいインスタンスへの接続に使用されます。詳細については、「Amazon EC2 のキーペアの使用」を参照してください。

- 前述の EC2 キーペアのプライベートキーを含む PEM ファイルの名前。PEM ファイルは、インスタンスにリモートで接続する場合に使用されます。

インスタンスの起動

次のスニペットでは、EC2 インスタンスが起動します。

このトピックの最後付近で、スニペットが実際に使用されている例を確認できます。

```
//
// 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;
}
```

インスタンスのモニタリング

次のスニペットでは、インスタンスが「Pending」状態でなくなるまでインスタンスを監視します。

このトピックの最後あたりで、スニペットが実際に使用されている例を確認できます。

Instance.State.Code プロパティの有効な値については、InstanceState クラスを参照してください。

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

コード全文

このセクションでは、例に関連する参考資料とコードの全文を示します。

SDK リファレンス

NuGet パッケージ:

- AWSSDK.EC2

プログラミング要素:

- 名前空間 Amazon.EC2

  クラス AmazonEC2Client

  クラス InstanceType
- 名前空間 Amazon.EC2.Model

  クラス DescribeInstancesRequest

  クラス DescribeInstancesResponse

  クラス Instance

  クラス InstanceNetworkInterfaceSpecification

クラス RunInstancesRequest

クラス RunInstancesResponse

コード

```
using System;
using System.Threading;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2LaunchInstance
{
  // = = =
```

```
         || (!string.IsNullOrEmpty(subnetID) && !subnetID.StartsWith("subnet-")))
        CommandLine.ErrorExit(
          "\nOne or more of the required arguments is missing or incorrect." +
          "\nRun the command with no arguments to see help.");

      // Create an EC2 client
      var ec2Client = new AmazonEC2Client();

      // Create an object with the necessary properties
      RunInstancesRequest request = GetRequestData(groupID, ami, keyPairName,
subnetID);

      // 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.");
    }
 }


 // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = =
 // 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);
        }
    }

}
```

その他の考慮事項

- EC2 インスタンスの状態をチェックするときに、DescribeInstancesRequest オブジェクトの `Filter` プロパティにフィルターを追加できます。この方法を使用すると、リクエストを特定のインスタンス (ユーザーが指定した特定のタグを持つインスタンスなど) に制限できます。

- 時間短縮のために、いくつかのプロパティには代表的な値が与えられています。これらのプロパティの一部またはすべてを、プログラムまたはユーザー入力で置き換えることができます。

- RunInstancesRequest オブジェクトの `MinCount` と `MaxCount` プロパティに使用できる値は、ターゲットのアベイラビリティーゾーンと、対象のインスタンスタイプに対して許可されているインスタンスの最大数によって決まります。詳細については、「Amazon EC2 よくある質問」の「Amazon EC2 ではいくつのインスタンスを実行できますか?」を参照してください。

- この例とは異なるインスタンスタイプを使用する場合は、いくつかのインスタンスタイプから選択できます。詳細については、「Amazon EC2 ユーザーガイド」の「Amazon EC2 インスタンスタイプ」を参照してください。「インスタンスタイプの詳細」と「インスタンスタイプエクスプローラー」も参照してください。

- インスタンスの起動時に、インスタンスに IAM ロールをアタッチすることもできます。そのためには、`Name` プロパティに IAM ロールの名前が設定された IamInstanceProfileSpecification オブジェクトを作成します。次に、そのオブジェクトを RunInstancesRequest オブジェクトの `IamInstanceProfile` プロパティに追加します。

  > **Note**
  >
  > IAM ロールが添付された EC2 インスタンスを起動するには、IAM ユーザーの設定に特定のアクセス許可が含まれている必要があります。必要なアクセス許可の詳細については、

「Amazon EC2 ユーザーガイド」の「IAM ロールをインスタンスに渡すアクセス許可をユーザーに付与する」を参照してください。

(オプション) インスタンスへの接続

インスタンスが実行状態になったら、適切なリモートクライアントを使用してインスタンスにリモート接続できます。Linux インスタンスと Windows インスタンスのいずれの場合も、インスタンスのパブリック IP アドレスまたは公開 DNS 名が必要です。また、以下も必要になります。

Linux インスタンスの場合

SSH クライアントを使用して Linux インスタンスに接続できます。「セキュリティグループの更新」の説明に従って、インスタンスの起動時に使用したセキュリティグループで、ポート 22 での SSH トラフィックが許可されていることを確認します。

また、インスタンスの起動に使用したキーペアのプライベート部分、すなわち PEM ファイルも必要です。

詳細については、「Amazon EC2 ユーザーガイド」の「Linux インスタンスへの接続」を参照してください。

Windows インスタンスの場合

RDP クライアントを使用してお使いのインスタンスに接続できます。「セキュリティグループの更新」の説明に従って、インスタンスの起動時に使用したセキュリティグループで、ポート 3389 での RDP トラフィックが許可されていることを確認します。

また、管理者パスワードも必要です。取得するには、以下のコード例を使用します。その際、インスタンス ID とインスタンスの起動に使用されたキーペアのプライベート部分、すなわち PEM ファイルが必要です。

詳細については、「Amazon EC2 ユーザーガイド」の「Windows インスタンスに接続する」を参照してください。

⚠ Warning

このサンプルコードは、インスタンスの管理者パスワードをプレーンテキストで返します。

SDK リファレンス

NuGet パッケージ:

- AWSSDK.EC2

プログラミング要素:

- 名前空間 Amazon.EC2

  クラス AmazonEC2Client
- 名前空間 Amazon.EC2.Model

  クラス GetPasswordDataRequest

  クラス GetPasswordDataResponse

コード

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2GetWindowsPassword
{
  // = = =
```

```
      // Get the application arguments from the parsed list
      string instanceID =
        CommandLine.GetArgument(parsedArgs, null, "-i", "--instance-id");
      string pemFileName =
        CommandLine.GetArgument(parsedArgs, null, "-p", "--pem-filename");
      if(   (string.IsNullOrEmpty(instanceID) || !instanceID.StartsWith("i-"))
         || (string.IsNullOrEmpty(pemFileName) || !pemFileName.EndsWith(".pem")))
        CommandLine.ErrorExit(
          "\nOne or more of the required arguments is missing or incorrect." +
          "\nRun the command with no arguments to see help.");

      // 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.");
    }
  }

  // = = =
```

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

```
}
```

クリーンアップ

EC2 インスタンスが不要になった場合は、「Amazon EC2 インスタンスの終了」の説明に従って必ずインスタンスを終了してください。

Amazon EC2 インスタンスの終了

Amazon EC2 インスタンスが必要なくなったときは、それらを終了できます。

この例では、 AWS SDK for .NET を使用して EC2 インスタンスを終了する方法を示します。アプリケーションは入力としてインスタンス ID を受け取ります。

SDK リファレンス

NuGet パッケージ:

- AWSSDK.EC2

プログラミング要素:

- 名前空間 Amazon.EC2

  クラス AmazonEC2Client
- 名前空間 Amazon.EC2.Model

  クラス TerminateInstancesRequest

  クラス 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);
      }
    }
  }
}
```

この例を実行した後、Amazon EC2 コンソールにサインインして EC2 インスタンスの終了を確認することをお勧めします。

Amazon EC2 スポットインスタンスのチュートリアル

このチュートリアルでは、 AWS SDK for .NET を使用して Amazon EC2 スポットインスタンスを管理する方法を示します。

概要:

スポットインスタンスでは、未使用の Amazon EC2 コンピューティング性能をオンデマンド料金よりも低価格でリクエストできます。これにより、中断が可能なアプリケーションの EC2 コストを大幅に削減できます。

スポットインスタンスのリクエスト方法と使用方法の概要は次のとおりです。

1. 支払う上限価格を指定して、スポットインスタンスリクエストを作成します。
2. リクエストが受理されたら、他の Amazon EC2 インスタンスと同様にインスタンスを実行します。
3. スポット料金の変更によってインスタンスが自動的に終了しない限り、必要な期間インスタンスを実行し、終了します。
4. 不要になったスポットインスタンスリクエストをクリーンアップして、スポットインスタンスがそれ以上作成されないようにします。

以上が、スポットインスタンスに関する非常に大まかな概要です。スポットインスタンスの詳細については、「Amazon EC2 ユーザーガイド」の「スポットインスタンス」を参照してください。

このチュートリアルの内容

このチュートリアルに従うときは、 AWS SDK for .NET を使用して以下を実行します。

- スポットインスタンスリクエストを作成する
- スポットインスタンスリクエストが受理されたかどうかを判断する
- スポットインスタンスリクエストをキャンセルする
- 関連するインスタンスを終了させる

以下のセクションでは、この例のスニペットとその他の情報を確認できます。スニペットの下には、この例のコードの全文が示されており、そのままビルドして実行できます。

トピック

- 前提条件
- 必要な要素を集める
- スポットインスタンスリクエストの作成
- スポットインスタンスリクエストの状態を判断する
- スポットインスタンスリクエストのクリーンアップ
- スポットインスタンスのクリーンアップ
- コード全文
- その他の考慮事項

前提条件

API の詳細と前提条件については、親セクション (Amazon EC2 の使用) を参照してください。

必要な要素を集める

スポットインスタンスリクエストを作成するには、いくつかの要素が必要です。

- インスタンスの数とそのインスタンスタイプ。選択できるインスタンスタイプは複数あります。詳細については、「Amazon EC2 ユーザーガイド」の「Amazon EC2 インスタンスタイプ」を参照してください。「インスタンスタイプの詳細」と「インスタンスタイプエクスプローラー」も参照してください。

  このチュートリアルでは、デフォルトの数は 1 です。

- インスタンスの作成に使用する Amazon マシンイメージ (AMI)。AMI の詳細については、「Amazon EC2 ユーザーガイド」の「Amazon マシンイメージ (AMI)」を参照してください。特に、「AMI の検索」と「共有 AMI」を参照してください。

- インスタンス時間あたりに支払う上限価格。すべてのインスタンスタイプ (オンデマンドインスタンスとスポットインスタンスの両方) の料金は、Amazon EC2 の料金ページで確認いただけます。このチュートリアルのデフォルト料金については、後で説明します。

- インスタンスにリモートで接続する場合は、適切な構成とリソースを持つセキュリティグループ。この詳細については「Amazon EC2 でのセキュリティグループの使用」で説明されており、必要な要素の収集とインスタンスへの接続に関する情報は「Amazon EC2 インスタンスの起動」で説

明されています。簡単にするために、このチュートリアルではすべての新しい AWS アカウントに存在する default という名前のセキュリティグループを使用します。

スポットインスタンスをリクエストするには複数の方法があります。一般的な戦略は以下のとおりです。

- オンデマンド料金を明確に下回るようにリクエストを作成します。
- 計算結果の価値に基づいてリクエストを作成します。
- コンピューティング性能をできるだけ早く獲得するようにリクエストを作成します。

以下の説明は、「Amazon EC2 ユーザーガイド」の「スポットインスタンスの料金履歴」を示しています。

コストをオンデマンドよりも削減する

実行完了までに何時間も、あるいは何日間もかかるバッチ処理ジョブがあるとします。ただし、いつ開始していつ終了するかについては、特に決められていないものとします。このジョブを完了するためのコストを、オンデマンドインスタンスを使用する場合よりも低くできるかどうかを考えます。

Amazon EC2 コンソールまたは Amazon EC2 API を使用して、インスタンスタイプ別のスポット料金の履歴を調べます。使用したいインスタンスタイプの、特定のアベイラビリティーゾーンでの価格履歴を分析した後は、リクエストのアプローチとして次の 2 つも考えられます。

- スポット料金の範囲の上限 (ただしオンデマンド料金よりは下) でリクエストを指定します。こうすることで、この 1 回限りのスポットリクエストが受理されて、ジョブが完了するまで連続して実行される可能性が高くなります。
- 価格範囲の下限でリクエストを指定し、1 つの永続リクエストで次々とインスタンスを起動するよう計画を立てます。これらのインスタンスの実行時間を合計すると、ジョブを完了するのに十分な長さとなり、合計コストも低くなります。

結果の価値以上は支払わない

データ処理ジョブを実行するとします。このジョブの結果が持つ価値は判明しており、計算コストに換算してどれくらいになるかもわかっています。

使用するインスタンスタイプのスポット料金履歴の分析が完了したら、計算時間のコストがこのジョブの結果の価値を上回ることのないように料金を選択します。永続リクエストを作成し、スポット料金がリクエスト以下となったときに断続的に実行するよう設定します。

コンピューティング性能をすぐに獲得する

追加のコンピューティング性能が突然、短期間だけ必要になり、オンデマンドインスタンスではそのコンピューティング性能を確保できないとします。使用するインスタンスタイプのスポット料金履歴の分析が完了したら、履歴での最高料金を超える料金を選択します。こうすることで、リクエストがすぐに受理され、計算が完了するまで連続して計算できる可能性が飛躍的に高まります。

必要な要素を集めて戦略を選択したら、スポットインスタンスをリクエストする準備ができました。このチュートリアルでは、デフォルトの最大スポットインスタンス料金をオンデマンド料金と同額 (このチュートリアルでは 0.003 ドル) に設定します。このように料金を設定すると、リクエストが達成される可能性が最大になります。

スポットインスタンスリクエストの作成

次のスニペットでは、上記で集めた要素を使用してスポットインスタンスリクエストを作成する方法を示しています。

このトピックの最後で、スニペットが実際に使用されている例を確認できます。

```
//
// 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];
}
```

このメソッドから返される重要な値は、スポットインスタンスリクエスト ID です。これは返された SpotInstanceRequest オブジェクトの SpotInstanceRequestId メンバーに含まれています。

> **Note**
>
> 起動したすべてのスポットインスタンスに対して料金が発生します。不要なコストを回避するには、リクエストをキャンセルしてインスタンスを削除するようにしてください。

スポットインスタンスリクエストの状態を判断する

次のスニペットでは、スポットインスタンスリクエストに関する情報を取得する方法を示しています。この情報を使用して、スポットインスタンスリクエストが受理されるのを待つかどうかといった特定の決定をコード内で下すことができます。

このトピックの最後で、スニペットが実際に使用されている例を確認できます。

```
//
// 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];
}
```

このメソッドでは、スポットインスタンスリクエストに関する情報 (インスタンス ID、状態、ステータスコードなど) が返されます。スポットインスタンスリクエストのステータスコードの詳細については、「Amazon EC2 ユーザーガイド」の「スポットリクエストのステータス」を参照してください。

スポットインスタンスリクエストのクリーンアップ

スポットインスタンスをリクエストする必要がなくなった場合、未処理のリクエストをキャンセルして、リクエストが再び受理されないようにすることが重要です。次のスニペットでは、スポットインスタンスリクエストをキャンセルする方法を示しています。

このトピックの最後で、スニペットが実際に使用されている例を確認できます。

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

スポットインスタンスのクリーンアップ

不要なコストを回避するには、スポットインスタンスリクエストから起動したインスタンスをすべて終了させることが重要です。単にスポットインスタンスリクエストをキャンセルするだけではインスタンスは終了しないので、引き続きインスタンスに対して料金が発生することになります。次のスニペットでは、アクティブなスポットインスタンスのインスタンス識別子を取得した後にインスタンスを終了する方法を示しています。

このトピックの最後で、スニペットが実際に使用されている例を確認できます。

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

コード全文

次のコード例では、前述のメソッドを呼び出して、スポットインスタンスリクエストを作成およびキャンセルし、スポットインスタンスを終了しています。

SDK リファレンス

NuGet パッケージ:

- AWSSDK.EC2

プログラミング要素:

- 名前空間 Amazon.EC2

  クラス AmazonEC2Client

  クラス InstanceType

- 名前空間 Amazon.EC2.Model

  クラス CancelSpotInstanceRequestsRequest

  クラス DescribeSpotInstanceRequestsRequest

  クラス DescribeSpotInstanceRequestsResponse

クラス InstanceStateChange

クラス LaunchSpecification

クラス RequestSpotInstancesRequest

クラス RequestSpotInstancesResponse

クラス SpotInstanceRequest

クラス TerminateInstancesRequest

クラス TerminateInstancesResponse

コード

```
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");
        }
      }
    }
  }
}
```

その他の考慮事項

- チュートリアルを実行したら、Amazon EC2 コンソールにサインインして、スポットインスタンスリクエストがキャンセル済みでスポットインスタンスが終了していることを確認するようお勧めします。

を使用した AWS Identity and Access Management (IAM) へのアクセス AWS SDK for .NET

は AWS SDK for .NET をサポートしています。これはAWS Identity and Access Management、AWS お客様がユーザーとユーザーのアクセス許可を管理できるようにするウェブサービスです AWS。

AWS Identity and Access Management (IAM) ユーザーは、ユーザーが作成するエンティティです AWS。エンティティは、 がやり取りするユーザーまたはアプリケーションを表します AWS。IAM ユーザーの詳細については、IAM ユーザーガイドの「IAM ユーザー」および「IAM と STS の制限」を参照してください。

IAM ポリシーを作成することによって、ユーザーにアクセス許可を付与します。ポリシーには、ユーザーが実行できるアクションと、そのアクションによって影響を受けるリソースの一覧が記載されたポリシードキュメントが含まれています。IAM ポリシーの詳細については、IAM ユーザーガイドの「ポリシーとアクセス許可」を参照してください。

> **Warning**
>
> セキュリティリスクを避けるため、専用ソフトウェアを開発するときや実際のデータを扱うときは、IAM ユーザーを認証に使用しないでください。代わりに、AWS IAM アイデンティティセンター などの ID プロバイダーとのフェデレーションを使用してください。

API

AWS SDK for .NET は、IAM クライアント用の APIs を提供します。API を使用すると、ユーザー、ロール、アクセスキーなどの IAM の機能を操作できます。

このセクションでは、これらの API を操作する際に活用できるパターンを示すいくつかの例を紹介します。API の完全なセットを確認するには、AWS SDK for .NET API リファレンスを参照してください (「Amazon.IdentityManagement」までスクロールします)。

このセクションには、認証情報の管理を容易にするために IAM ロールを Amazon EC2 インスタンスにアタッチする方法を示す例も含まれています。

IAM API は、AWSSDK.IdentityManagement NuGet パッケージで提供されます。

前提条件

開始する前に、環境をセットアップし、プロジェクトを設定したことを確認してください。また、「SDK の使用」の情報を確認してください。

トピック

トピック

- JSON からの IAM マネージドポリシーの作成
- IAM マネージドポリシーのポリシードキュメントの表示
- IAM ロールを使用したアクセス権の付与

JSON からの IAM マネージドポリシーの作成

この例では、 AWS SDK for .NET を使用して、JSON の特定のポリシードキュメントから IAM 管理ポリシーを作成する方法を示します。アプリケーションは IAM クライアントオブジェクトを作成し、ファイルからポリシードキュメントを読み取ってポリシーを作成します。

> **Note**
>
> JSON 形式のポリシードキュメントの例については、このトピックの最後にあるその他の考慮事項を参照してください。

以下のセクションでは、この例のスニペットを確認できます。その下には、この例のコードの全文が示されており、そのままビルドして実行できます。

トピック

- ポリシーの作成
- コード全文
- その他の考慮事項

ポリシーの作成

次のスニペットでは、指定された名前とポリシードキュメントを使用して IAM マネージドポリシーを作成します。

このトピックの最後で、スニペットが実際に使用されている例を確認できます。

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

コード全文

このセクションでは、例に関連する参考資料とコードの全文を示します。

SDK リファレンス

NuGet パッケージ:

- AWSSDK.IdentityManagement

プログラミング要素:

- 名前空間 Amazon.IdentityManagement

  クラス AmazonIdentityManagementServiceClient
- 名前空間 Amazon.IdentityManagement.Model

  クラス CreatePolicyRequest

  クラス CreatePolicyResponse

コード

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Amazon.IdentityManagement;
using Amazon.IdentityManagement.Model;

namespace IamCreatePolicyFromJson
{
  // = = =
```

```
      string policyName =
        CommandLine.GetArgument(parsedArgs, null, "-p", "--policy-name");
      string policyFilename =
        CommandLine.GetArgument(parsedArgs, null, "-j", "--json-filename");
      if(   string.IsNullOrEmpty(policyName)
         || (string.IsNullOrEmpty(policyFilename) || !
policyFilename.EndsWith(".json")))
        CommandLine.ErrorExit(
          "\nOne or more of the required arguments is missing or incorrect." +
          "\nRun the command with no arguments to see help.");

      // Create an IAM service client
      var iamClient = new AmazonIdentityManagementServiceClient();

      // Create the new policy
      var response = await CreateManagedPolicy(iamClient, policyName, policyFilename);
      Console.WriteLine($"\nPolicy {response.Policy.PolicyName} has been created.");
      Console.WriteLine($"  Arn: {response.Policy.Arn}");
    }


    //
    // 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)});
    }


    //
    // 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.");
    }
  }
```

```
 // = = =
```

```
          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);
    }
  }

}
```

その他の考慮事項

- 次に示すポリシードキュメントの例を JSON ファイルにコピーして、このアプリケーションの入力として使用できます。

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": "*"
    }
  ]
}
```

- ポリシーが作成されたことを IAM コンソールで確認できます。[Filter policies] (フィルターポリシー) ドロップダウンリストで、[Customer managed] (カスタマー管理) を選択します。ポリシーが不要になった場合は削除します。

- ポリシー作成の詳細については、IAM ユーザーガイドの「IAM ポリシーの作成」と「IAM JSON ポリシーリファレンス」を参照してください。

IAM マネージドポリシーのポリシードキュメントの表示

この例では、 AWS SDK for .NET を使用してポリシードキュメントを表示する方法を示します。アプリケーションは IAM クライアントオブジェクトを作成し、指定された IAM マネージドポリシーのデフォルトバージョンを検索して、JSON 形式のポリシードキュメントを表示します。

以下のセクションでは、この例のスニペットを確認できます。その下には、この例のコードの全文が示されており、そのままビルドして実行できます。

トピック

- デフォルトバージョンの検索
- ポリシードキュメントの表示
- コード全文

デフォルトバージョンの検索

次のスニペットでは、指定された IAM ポリシーのデフォルトバージョンを検索します。

このトピックの最後で、スニペットが実際に使用されている例を確認できます。

```
//
// 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;
}
```

ポリシードキュメントの表示

次のスニペットでは、指定された IAM ポリシーの JSON 形式のポリシードキュメントを表示します。

このトピックの最後で、スニペットが実際に使用されている例を確認できます。

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

コード全文

このセクションでは、例に関連する参考資料とコードの全文を示します。

SDK リファレンス

NuGet パッケージ:

- AWSSDK.IdentityManagement

プログラミング要素:

- 名前空間 Amazon.IdentityManagement

  クラス AmazonIdentityManagementServiceClient
- 名前空間 Amazon.IdentityManagement.Model

クラス GetPolicyVersionRequest

クラス GetPolicyVersionResponse

クラス ListPolicyVersionsRequest

クラス ListPolicyVersionsResponse

クラス PolicyVersion

コード

```
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)}");
      }
    }
}
```

IAM ロールを使用したアクセス権の付与

このチュートリアルでは、 AWS SDK for .NET を使用して Amazon EC2 インスタンスで IAM ロールを有効にする方法を示します。

概要:

へのすべてのリクエストは、 によって発行された認証情報を使用して暗号化して署名 AWS する必要があります AWS。したがって、Amazon EC2 インスタンスで実行するアプリケーション用の認証情報を管理するための戦略が必要です。これらの認証情報を、アプリケーションからのアクセスを維持したまま安全に配信、保存、ローテーションする必要があります。

IAM ロールを使用すると、認証情報を効果的に管理できます。IAM ロールを作成し、アプリケーションに必要なアクセス許可を使用して設定を行った後で、そのロールを EC2 インスタンスにアタッチします。IAM ロールを使用する利点の詳細については、「Amazon EC2 ユーザーガイド」の「Amazon EC2 の IAM ロール」を参照してください。また、IAM ユーザーガイドの「IAM ロール」の情報も参照してください。

を使用して構築されたアプリケーションの場合 AWS SDK for .NET、アプリケーションが AWS サービスのクライアントオブジェクトを構築すると、オブジェクトはいくつかの潜在的なソースから認証情報を検索します。検索の順序は、「認証情報とプロファイルの解決」に示されています。

クライアントオブジェクトが他のソースから認証情報を見つけられなかった場合、クライアントオブジェクトは IAM ロールに設定された、EC2 インスタンスのメタデータにあるものと同じアクセス許可を持つ一時的な認証情報を取得します。これらの認証情報は、クライアントオブジェクト AWS から を呼び出すために使用されます。

このチュートリアルの内容

このチュートリアルでは、 AWS SDK for .NET (およびその他のツール) を使用して IAM ロールがアタッチされた Amazon EC2 インスタンスを起動し、IAM ロールのアクセス許可を使用してインスタンス上のアプリケーションを確認します。

トピック

- サンプルの Amazon S3 アプリケーションの作成
- IAM ロールの作成
- EC2 インスタンスの起動と IAM ロールのアタッチ
- EC2 インスタンスへの接続
- EC2 インスタンスでのサンプルアプリケーションの実行
- クリーンアップ

サンプルの Amazon S3 アプリケーションの作成

このサンプルアプリケーションは、Amazon S3 からオブジェクトを取得します。アプリケーションを実行するには、以下が必要です。

- テキストファイルを含む Amazon S3 バケット。
- AWS バケットへのアクセスを許可する開発マシンの 認証情報。

Amazon S3 バケットの作成およびオブジェクトのアップロードの詳細については、「Amazon Simple Storage Service ユーザーガイド」を参照してください。 AWS 認証情報の詳細については、「」を参照してくださいAWS SDK for .NET を使用した の認証 AWS。

以下のコードを使用して、.NET Core プロジェクトを作成します。次に、開発マシンでアプリケーションをテストします。

> **Note**
>
> 開発マシンには .NET Core Runtime がインストールされており、アプリケーションを公開しなくても実行できます。このチュートリアルの後半で EC2 インスタンスを作成するときに、インスタンスに .NET Core Runtime をインストールすることができます。こうすることで同様のエクスペリエンスを実現でき、ファイル転送量が小さくなります。

ただし、インスタンスに .NET Core Runtime をインストールしない選択も可能です。この場合は、インスタンスを転送するときにすべての依存関係が含まれるようにアプリケーションを公開する必要があります。

SDK リファレンス

NuGet パッケージ:

- AWSSDK.S3

プログラミング要素:

- 名前空間 Amazon.S3

  クラス AmazonS3Client
- 名前空間 Amazon.S3.Model

  クラス GetObjectResponse

コード

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

namespace S3GetTextItem
{
  // = = =
```

```
    {
      PrintHelp();
      return;
    }

    // Get the application arguments from the parsed list
    string bucket =
      CommandLine.GetArgument(parsedArgs, null, "-b", "--bucket-name");
    string item =
      CommandLine.GetArgument(parsedArgs, null, "-t", "--text-object");
    string outFile =
      CommandLine.GetArgument(parsedArgs, null, "-o", "--output-filename");
    if(   string.IsNullOrEmpty(bucket)
       || string.IsNullOrEmpty(item)
       || string.IsNullOrEmpty(outFile))
      CommandLine.ErrorExit(
        "\nOne or more of the required arguments is missing or incorrect." +
        "\nRun the command with no arguments to see help.");

    // Create the S3 client object and get the file object from the bucket.
    var response = await GetObject(new AmazonS3Client(), bucket, item);

    // Write the contents of the file object to the given output file.
    var reader = new StreamReader(response.ResponseStream);
    string contents = reader.ReadToEnd();
    using (var s = new FileStream(outFile, FileMode.Create))
    using (var writer = new StreamWriter(s))
      writer.WriteLine(contents);
  }


  //
  // Method to get an object from an S3 bucket.
  private static async Task<GetObjectResponse> GetObject(
    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.");
    }
  }


  // = = =
```

```
          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);
    }
  }
```

```
}
```

必要に応じて、開発マシンで使用している認証情報を一時的に削除して、アプリケーションの応答を確認できます。(ただし、完了したら認証情報を忘れずに復元してください)

IAM ロールの作成

Amazon S3 にアクセスするための適切なアクセス許可を持つ IAM ロールを作成します。

1. [IAM コンソール] を開きます。
2. ナビゲーションペインで [Roles] (ロール) を選択し、続いて [Create Role] (ロールの作成) を選択します。
3. [AWS サービス] を選択し、[EC2] を見つけて選択して、[次へ: アクセス許可] を選択します。
4. [Attach permissions policies] (アクセス許可ポリシーの添付) をクリックし、[AmazonS3ReadOnlyAccess] を見つけて選択します。必要に応じてポリシーを確認し、[Next: Tags] (次へ: タグ) を選択します。
5. 必要に応じてタグを追加し、[Next: Review] (次へ: レビュー) を選択します。
6. ロールの名前と説明を入力し、[Create role] (ロールの作成) を選択します。この名前は EC2 インスタンスを起動するときに必要になるため、忘れないでください。

EC2 インスタンスの起動と IAM ロールのアタッチ

先ほど作成した IAM ロールを使用して EC2 インスタンスを起動します。これは以下の方法で実行できます。

- EC2 コンソールの使用

  EC2 コンソールを使用してインスタンスを起動するには、「Amazon EC2 ユーザーガイド」の「新しいインスタンス起動ウィザードを使用してインスタンスを起動する」を参照してください。

  起動ページを見ながら、少なくとも [詳細] ペインを展開して、[IAM インスタンスプロファイル] で以前に作成した IAM ロールを指定できます。

- の使用 AWS SDK for .NET

  詳細については、「Amazon EC2 インスタンスの起動」を参照してください (トピックの最後近くにある「その他の考慮事項」を含む)。

IAM ロールが添付された EC2 インスタンスを起動するには、IAM ユーザーの設定に特定のアクセス許可が含まれている必要があります。必要なアクセス許可の詳細については、「Amazon EC2 ユーザーガイド」の「IAM ロールをインスタンスに渡すアクセス許可をユーザーに付与する」を参照してください。

EC2 インスタンスへの接続

EC2 インスタンスに接続することで、サンプルアプリケーションをそのインスタンスに転送して、アプリケーションを実行できるようにします。また、インスタンスの起動に使用したキーペアのプライベート部分を含むファイル、すなわち PEM ファイルも必要です。

Linux インスタンスに接続する方法については、「Amazon EC2 ユーザーガイド」の「Linux インスタンスに接続する」または「Windows インスタンスに接続する」を参照してください。接続する際は、開発マシンからインスタンスにファイルを転送できるように接続してください。

Windows で Visual Studio を使用している場合は、Toolkit for Visual Studio を使用してインスタンスに接続することもできます。詳細については、「 AWS Toolkit for Visual Studio ユーザーガイド」の Amazon EC2 インスタンスへの接続」を参照してください。

EC2 インスタンスでのサンプルアプリケーションの実行

1. ローカルドライブからインスタンスにアプリケーションファイルをコピーします。

   転送するファイルは、アプリケーションのビルド方法と、インスタンスに .NET Core Runtime がインストールされているかどうかによって異なります。インスタンスにファイルを転送する方法については、「Amazon EC2 ユーザーガイド」の「Linux インスタンスに接続する」(該当するサブセクションを参照) または「Windows インスタンスにファイルを転送する」を参照してください。
2. アプリケーションを起動し、開発マシンと同じ実行結果が得られることを確認します。
3. アプリケーションで、IAM ロールによって提供されている認証情報が使用されていることを確認します。

   a. Amazon EC2 コンソールを開きます。

   b. インスタンスを選択し、[Actions] (アクション)、[Instance Settings] (インスタンスの設定)、[Attach/Replace IAM Role] (IAM ロールの添付/置換) を使用して IAM ロールをデタッチします。

   c. アプリケーションを再度実行して、認可エラーが返されることを確認します。

クリーンアップ

チュートリアルが終了し、作成した EC2 インスタンスが不要になった場合は、不要な料金が発生しないようにインスタンスを終了してください。この操作は Amazon EC2 コンソール で、または Amazon EC2 インスタンスの終了 の説明に従ってプログラムを使用して実行できます。必要に応じて、このチュートリアル用に作成した他のリソースも削除できます。削除可能なリソースには、IAM ロール、EC2 キーペアと PEM ファイル、セキュリティグループなどがあります。

Amazon Simple Storage Service インターネットストレージの使用

は、インターネット用のストレージである Amazon S3 AWS SDK for .NET をサポートしています。Web スケールのコンピューティングを開発者が容易にできるように設計されています。

API

AWS SDK for .NET はAPIs Amazon S3を提供します。API を使用すると、バケットやアイテムなどの Amazon S3 リソースを操作できます。Amazon S3 向け API の完全なセットを表示するには、以下を参照してください。

- AWS SDK for .NET API リファレンス (「Amazon.S3」までスクロールします)。
- Amazon.Extensions.S3.Encryption のドキュメント

Amazon S3 API は、以下の NuGet パッケージによって提供されます。

- AWSSDK.S3
- Amazon.Extensions.S3.Encryption

前提条件

開始する前に、環境をセットアップし、プロジェクトを設定したことを確認してください。また、「SDK の使用」の情報を確認してください。

このドキュメントで取り上げる例

このドキュメントの以下のトピックでは、 AWS SDK for .NET を使用して Amazon S3 を操作する方法について説明します。

- S3 暗号化用 KMS キーの使用

他のドキュメントで取り上げられている例

以下の Amazon S3 デベロッパーガイドへのリンクは、 を使用して Amazon S3 AWS SDK for .NET を操作する方法の追加の例を示しています。

> **Note**
>
> これらの例と追加のプログラミング上の考慮事項は、.NET Framework AWS SDK for .NET を使用する のバージョン 3 用に作成されていますが、.NET Core AWS SDK for .NET を使用する のそれ以降のバージョンでも実行可能です。コードの軽微な修正が必要になる場合があります。

Amazon S3 プログラミングの例

- ACL の管理
- バケットの作成
- オブジェクトのアップロード
- 高レベル API でのマルチパートアップロード (Amazon.S3.Transfer.TransferUtility)
- 低レベル API でのマルチパートアップロード
- オブジェクトのリスト作成
- キーのリスト表示
- オブジェクトの取得
- オブジェクトのコピー
- マルチパートアップロード API を使用したオブジェクトのコピー
- オブジェクトの削除
- 複数のオブジェクトの削除
- オブジェクトの復元
- 通知用のバケットの設定
- オブジェクトのライフサイクルを管理する
- 署名付きオブジェクト URL の生成
- ウェブサイトの管理
- Cross-Origin Resource Sharing (CORS) の有効化

プログラミングに関するその他の考慮事項

- Amazon S3 プログラミングでの AWS SDK for .NET の使用
- IAM ユーザーの一時的な認証情報を使用したリクエストの実行
- フェデレーションユーザーの一時的な認証情報を使用したリクエストの実行
- サーバー側暗号化の指定
- お客様が用意した暗号化キーを使用したサーバー側暗号化の指定

での Amazon S3 暗号化の AWS KMS キーの使用 AWS SDK for .NET

この例では、 AWS Key Management Service キーを使用して Amazon S3 オブジェクトを暗号化する方法を示します。アプリケーションはカスタマーマスターキー (CMK) を作成し、これを使用してクライアント側の暗号化に使用する AmazonS3EncryptionClientV2 オブジェクトを作成します。アプリケーションはそのクライアントを使用して、既存の Amazon S3 バケット内の所定のテキストファイルから暗号化されたオブジェクトを作成します。次にオブジェクトを復号化して、その内容を表示します。

> **Warning**
>
> `AmazonS3EncryptionClient` という類似のクラスは `AmazonS3EncryptionClientV2` クラスよりも安全性が低く、非推奨です。`AmazonS3EncryptionClient` を使用している既存のコードを移行するには、「S3 暗号化クライアントの移行 (V1 から V2)」を参照してください。

トピック

- 暗号化マテリアルの作成
- Amazon S3 オブジェクトの作成と暗号化
- コード全文
- その他の考慮事項

暗号化マテリアルの作成

次のスニペットでは、KMS キー ID を含む `EncryptionMaterials` オブジェクトを作成します。

このトピックの最後で、スニペットが実際に使用されている例を確認できます。

```
    // 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);
```

Amazon S3 オブジェクトの作成と暗号化

次のスニペットでは、先ほど作成した暗号化マテリアルを使用する AmazonS3EncryptionClientV2 オブジェクトを作成します。次に、クライアントを使用して新しい Amazon S3 オブジェクトを作成し、暗号化します。

このトピックの最後 で、スニペットが実際に使用されている例を確認できます。

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

```
    }
```

コード全文

このセクションでは、例に関連する参考資料とコードの全文を示します。

SDK リファレンス

NuGet パッケージ:

- Amazon.Extensions.S3.Encryption

プログラミング要素:

- 名前空間 Amazon.Extensions.S3.Encryption

  クラス AmazonS3EncryptionClientV2

  クラス AmazonS3CryptoConfigurationV2

  クラス CryptoStorageMode

  クラス EncryptionMaterialsV2
- 名前空間 Amazon.Extensions.S3.Encryption.Primitives

  クラス KmsType
- 名前空間 Amazon.S3.Model

  クラス GetObjectRequest

  クラス GetObjectResponse

  クラス PutObjectRequest
- 名前空間 Amazon.KeyManagementService

  クラス AmazonKeyManagementServiceClient
- 名前空間 Amazon.KeyManagementService.Model

  クラス CreateKeyRequest

  クラス CreateKeyResponse

コード

```
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
{
  // = = =
```

```
      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.");
    }

 }

 // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = =
 // 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);
    }
  }

}
```

その他の考慮事項

- この例の結果を確認できます。そのためには Amazon S3 コンソールに移動し、アプリケーションに対して指定したバケットを開きます。次に、新しいオブジェクトを見つけてダウンロードし、テキストエディタで開きます。

- AmazonS3EncryptionClientV2 クラスでは、標準の AmazonS3Client クラスと同じインターフェイスが実装されています。そのため、暗号化と復号化がクライアント側で自動的かつ透過的に行われるよう、コードを AmazonS3EncryptionClientV2 クラスに移植するのが容易になります。

- キーをマスター AWS KMS キーとして使用する利点の 1 つは、独自のマスターキーを保存および管理する必要がないことです。これは によって行われます AWS。もう 1 つの利点は、 の AmazonS3EncryptionClientV2 クラス AWS SDK for .NET が の AmazonS3EncryptionClientV2 クラスと相互運用できることです AWS SDK for Java。つまり、 で暗号化 AWS SDK for Java し AWS SDK for .NET、 で復号できます。逆も同様です。

  > **Note**
  >
  > の AmazonS3EncryptionClientV2 クラスは、メタデータモードで実行されている場合にのみ KMS マスターキー AWS SDK for .NET をサポートします。の AmazonS3EncryptionClientV2クラスの命令ファイルモード AWS SDK for .NET は、の AmazonS3EncryptionClientV2クラスの と互換性がありません AWS SDK for Java。

- `AmazonS3EncryptionClientV2` クラスによるクライアント側の暗号化、およびエンベロープ暗号化の仕組みの詳細については、「AWS SDK for .NET および Amazon S3 によるクライアント側のデータ暗号化」を参照してください。

Amazon Simple Notification Service を使用したクラウドからの通知の送信

> **Note**
>
> このトピックの情報は、.NET Framework および AWS SDK for .NET バージョン 3.3 以前のプロジェクトに固有のものです。

は、Amazon Simple Notification Service (Amazon SNS) AWS SDK for .NET をサポートしています。Amazon SNS は、アプリケーション、エンドユーザー、デバイスがクラウドから通知を即座に送信できるようにするウェブサービスです。詳細については、「Amazon SNS」を参照してください。

Amazon SNS トピックの一覧表示

次の例では、Amazon SNS トピック、各トピックのサブスクリプション、および各トピックの属性を一覧表示する方法を示します。この例では、デフォルトのオペレーティングシステムである AmazonSimpleNotificationServiceClient を使用します。

```
// 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));
```

Amazon SNS トピックへのメッセージの送信

次の例は、Amazon SNS トピックにメッセージを送信する方法を示します。この例では、1 つの引数 (Amazon SNS トピックの ARN) を取ります。

```
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);
            }
        }
    }
}
```

GitHub で、コマンドラインからサンプルをビルドして実行する方法に関する情報を含む、完全な例を参照してください。

1 つの電話番号に SMS メッセージを送信する

次の例は、電話番号に SMS メッセージを送信する方法を示します。この例では、1 つの引数 (電話番号) を取ります。この引数は、コメントに記載されている 2 つの形式のいずれかである必要があります。

```
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);
            }
        }
    }
}
```

GitHub で、コマンドラインからサンプルをビルドして実行する方法に関する情報を含む、完全な例を参照してください。

Amazon SQS を使用したメッセージング

は、Amazon Simple Queue Service (Amazon SQS) AWS SDK for .NET をサポートします。これは、システム内のコンポーネント間のメッセージまたはワークフローを処理するメッセージキューイングサービスです。

Amazon SQS キューは、マイクロサービス、分散システム、サーバーレスアプリケーションなどのソフトウェアコンポーネント間でメッセージを送信、保存、受信できる仕組みを提供します。その結果、こうしたコンポーネントを切り離すことができるようになり、独自のメッセージングシステムを設計および運用する必要がなくなります。Amazon SQS でのキューとメッセージの仕組みに関する詳細については、Amazon SQS のチュートリアルおよび「Amazon Simple Queue Service デベロッパーガイド」の「Amazon SQS の基本的なアーキテクチャ」を参照してください。

> **Important**
>
> キューが持つ分散的な性質のため、Amazon SQS ではメッセージを送信された順序で受信することは保証できません。メッセージの順序を保持する必要がある場合は、Amazon SQS FIFO キューを使用してください。

API

AWS SDK for .NET はAPIs Amazon SQS を提供します。API を使用すると、キューやメッセージなどの Amazon SQS 機能を操作できます。このセクションでは、これらの API を操作する際に活用できるパターンを示すいくつかの例を紹介します。API の完全なセットを確認するには、AWS SDK for .NET API リファレンスを参照してください (「Amazon.SQS」までスクロールします)。

Amazon SQS API は、AWSSDK.SQS NuGet パッケージによって提供されます。

前提条件

開始する前に、環境をセットアップし、プロジェクトを設定したことを確認してください。また、「SDK の使用」の情報を確認してください。

トピック

トピック

- Amazon SQS キューの作成
- Amazon SQS キューの更新
- Amazon SQS キューの削除
- Amazon SQS メッセージの送信
- Amazon SQS メッセージの受信

Amazon SQS キューの作成

この例では、 AWS SDK for .NET を使用して Amazon SQS キューを作成する方法を示します。ユーザーがデッドレターキューの ARN を指定しない場合、アプリケーションはデッドレターキューを作成します。次に、デッドレターキュー (ユーザーが指定したもの、または作成されたもの) を含む標準メッセージキューを作成します。

コマンドライン引数を何も指定しない場合、アプリケーションは単に既存のすべてのキューに関する情報を表示します。

以下のセクションでは、この例のスニペットを確認できます。その下には、この例のコードの全文が示されており、そのままビルドして実行できます。

トピック

- 既存のキューの表示

- キューの作成
- キューの ARN の取得
- コード全文
- その他の考慮事項

既存のキューの表示

次のスニペットでは、SQS クライアントのリージョンにある既存のキューのリストと、各キューの属性を表示します。

このトピックの最後で、スニペットが実際に使用されている例を確認できます。

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

キューの作成

次のスニペットでは、キューが作成されます。このスニペットにはデッドレターキューの使用が含まれていますが、デッドレターキューは必ずしもキューに必要ではありません。

このトピックの最後で、スニペットが実際に使用されている例を確認できます。

```
    //
    // 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;
    }
```

キューの ARN の取得

次のスニペットでは、指定されたキュー URL で特定されるキューの ARN を取得します。

このトピックの最後で、スニペットが実際に使用されている例を確認できます。

```
    //
    // 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;
    }
```

コード全文

このセクションでは、例に関連する参考資料とコードの全文を示します。

SDK リファレンス

NuGet パッケージ:

- AWSSDK.SQS

プログラミング要素:

- 名前空間 Amazon.SQS

  クラス AmazonSQSClient

  クラス QueueAttributeName
- 名前空間 Amazon.SQS.Model

  クラス CreateQueueRequest

  クラス CreateQueueResponse

  クラス GetQueueAttributesResponse

  クラス ListQueuesResponse

コード

```
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.SQS;
using Amazon.SQS.Model;
```

```
namespace SQSCreateQueue
{
  // = = =
```

```
        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}.");
    }
  }


  // = = =
```

```
    // - 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);
    }
  }

}
```

その他の考慮事項

- キュー名は、英数字、ハイフン、およびアンダースコアで構成する必要があります。

- キュー名とキュー URL では大文字と小文字が区別されます。

- キュー URL が必要なときにキュー名しかない場合には、AmazonSQSClient.GetQueueUrlAsync メソッドのいずれかを使用してください。

- 設定できるさまざまなキュー属性の詳細については、AWS SDK for .NET API リファレンスの「CreateQueueRequest」か、または Amazon Simple Queue Service API リファレンスの「SetQueueAttributes」を参照してください。

- この例では、作成するキュー上のすべてのメッセージに対してロングポーリングを指定します。これは ReceiveMessageWaitTimeSeconds 属性を使用して行われます。

AmazonSQSClient クラスの `ReceiveMessageAsync` メソッドの呼び出し中にロングポーリングを指定することもできます。詳細については、「Amazon SQS メッセージの受信」を参照してください。

ショートポーリングとロングポーリングの違いに関する詳細については、Amazon Simple Queue Service デベロッパーガイドの「ショートポーリングとロングポーリング」を参照してください。

- デッドレターキューとは、他の (送信元) キューが正常に処理されないメッセージの送信先として使用できるキューのことです。詳細については、Amazon Simple Queue Service デベロッパーガイドの「Amazon SQS デッドレターキュー」を参照してください。

- キューのリストとこの例の結果は、Amazon SQS コンソールでも確認できます。

Amazon SQS キューの更新

この例では、 AWS SDK for .NET を使用して Amazon SQS キューを更新する方法を示します。アプリケーションはいくつかのチェックを行った後、指定された属性を指定された値で更新し、キューのすべての属性を表示します。

コマンドライン引数にキュー URL のみを含めた場合、アプリケーションは単にキューのすべての属性を表示します。

以下のセクションでは、この例のスニペットを確認できます。その下には、この例のコードの全文が示されており、そのままビルドして実行できます。

トピック

- キュー属性の表示
- 属性名の検証
- キュー属性の更新
- コード全文
- その他の考慮事項

キュー属性の表示

次のスニペットでは、指定されたキュー URL で特定されるキューの属性を表示します。

このトピックの最後で、スニペットが実際に使用されている例を確認できます。

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

属性名の検証

次のスニペットでは、更新される属性の名前を検証します。

このトピックの最後で、スニペットが実際に使用されている例を確認できます。

```
//
// 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;
}
```

キュー属性の更新

次のスニペットでは、指定されたキュー URL で特定されるキューの属性を更新します。

このトピックの最後で、スニペットが実際に使用されている例を確認できます。

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

コード全文

このセクションでは、例に関連する参考資料とコードの全文を示します。

SDK リファレンス

NuGet パッケージ:

- AWSSDK.SQS

プログラミング要素:

- 名前空間 Amazon.SQS

  クラス AmazonSQSClient

  クラス QueueAttributeName
- 名前空間 Amazon.SQS.Model

  クラス GetQueueAttributesResponse

コード

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSUpdateQueue
{
  // = = =
```

```
    private const int MaxArgs = 3;
    private const int InvalidArgCount = 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) || (parsedArgs.Count == InvalidArgCount))
        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 qUrl = CommandLine.GetArgument(parsedArgs, null, "-q");
      var attribute = CommandLine.GetArgument(parsedArgs, null, "-a");
      var value = CommandLine.GetArgument(parsedArgs, null, "-v", "--value");

      if(string.IsNullOrEmpty(qUrl))
        CommandLine.ErrorExit("\nYou must supply at least a queue URL." +
          "\nRun the command with no arguments to see help.");

      // 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.");
    }
  }


  // = = =
```

```
          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);
```

```
    }
  }

}
```

その他の考慮事項

- RedrivePolicy 属性を更新するには、値全体を引用符で囲み、キーと値のペアの引用符をお使いのオペレーティングシステムに応じた適切な方法でエスケープする必要があります。

  例えば Windows では、値は次のような形で構成されます。

  ```
  "{\"deadLetterTargetArn\":\"DEAD_LETTER-QUEUE-ARN\",\"maxReceiveCount\":\"10\"}"
  ```

Amazon SQS キューの削除

この例では、 AWS SDK for .NET を使用して Amazon SQS キューを削除する方法を示します。アプリケーションはキューを削除し、キューがなくなるまで一定時間待ってから、残っているキューのリストを表示します。

コマンドライン引数を何も指定しない場合、アプリケーションは単に既存のキューのリストを表示します。

以下のセクションでは、この例のスニペットを確認できます。その下には、この例のコードの全文が示されており、そのままビルドして実行できます。

トピック

- キューの削除
- キューがなくなるまで待機
- 既存のキューのリストの表示
- コード全文
- その他の考慮事項

キューの削除

次のスニペットでは、指定されたキュー URL で特定されるキューを削除します。

このトピックの最後で、スニペットが実際に使用されている例を確認できます。

```
    //
    // 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.");
    }
```

キューがなくなるまで待機

次のスニペットでは、削除プロセスが完了するまで待機します。この処理には 60 秒かかる場合があります。

このトピックの最後で、スニペットが実際に使用されている例を確認できます。

```
    //
    // 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;
      }
```

```
    }
```

既存のキューのリストの表示

次のスニペットでは、SQS クライアントのリージョンにある既存のキューのリストを表示します。

このトピックの最後で、スニペットが実際に使用されている例を確認できます。

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

コード全文

このセクションでは、例に関連する参考資料とコードの全文を示します。

SDK リファレンス

NuGet パッケージ:

- AWSSDK.SQS

プログラミング要素:

- 名前空間 Amazon.SQS

  クラス AmazonSQSClient
- 名前空間 Amazon.SQS.Model

  クラス ListQueuesResponse

コード

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

```
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSDeleteQueue
{
  // = = =
```

```
        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}");
    }
  }
}
```

その他の考慮事項

- DeleteQueueAsync API コールでは、削除しようとしているキューがデッドレターキューとして使用されているかどうかはチェックされません。チェックするには、より高度な手順が必要になります。

- キューのリストとこの例の結果は、Amazon SQS コンソールでも確認できます。

Amazon SQS メッセージの送信

この例では、 を使用して Amazon SQS キューに AWS SDK for .NET メッセージを送信する方法を示します。このキューは、プログラムで作成することも、Amazon SQS コンソールを使用して作成することもできます。アプリケーションは 1 つのメッセージをキューに送信し、次にメッセージのバッチを送信します。その後、アプリケーションはユーザーからの入力を待ちます。入力としては、キューに送信する追加メッセージや、アプリケーションの終了要求などが考えられます。

この例と、次のメッセージの受信に関する例を一緒に使用して、Amazon SQS のメッセージフローを確認できます。

以下のセクションでは、この例のスニペットを確認できます。その下には、この例のコードの全文が示されており、そのままビルドして実行できます。

トピック

- メッセージの送信
- メッセージのバッチの送信
- キューからのすべてのメッセージの削除

- コード全文
- その他の考慮事項

メッセージの送信

次のスニペットでは、指定されたキュー URL で特定されるキューにメッセージを送信します。

このトピックの最後で、スニペットが実際に使用されている例を確認できます。

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

メッセージのバッチの送信

次のスニペットでは、指定されたキュー URL で特定されるキューにメッセージのバッチを送信します。

このトピックの最後で、スニペットが実際に使用されている例を確認できます。

```
//
// 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.");
}
```

キューからのすべてのメッセージの削除

次のスニペットでは、指定されたキュー URL で特定されるキューからすべてのメッセージを削除します。この操作は、キューの消去とも呼ばれます。

このトピックの最後で、スニペットが実際に使用されている例を確認できます。

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

コード全文

このセクションでは、例に関連する参考資料とコードの全文を示します。

SDK リファレンス

NuGet パッケージ:

- AWSSDK.SQS

プログラミング要素:

- 名前空間 Amazon.SQS

  クラス AmazonSQSClient
- 名前空間 Amazon.SQS.Model

  クラス PurgeQueueResponse

  クラス SendMessageBatchResponse

  クラス SendMessageResponse

  クラス SendMessageBatchRequestEntry

  クラス SendMessageBatchResultEntry

コード

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSSendMessages
{
  // = = =
```

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

その他の考慮事項

- 許可される文字など、メッセージのさまざまな制限については、Amazon Simple Queue Service デベロッパーガイドの「メッセージに関連するクォータ」を参照してください。

- メッセージは削除されるか、またはキューが消去されるまでキューに残ります。アプリケーションがメッセージを受信すると、そのメッセージはキュー内にまだ存在していたとしてもキューで表

示されなくなります。可視性タイムアウトの詳細については、「Amazon SQS 可視性タイムアウト」を参照してください。

- メッセージ本文に加えて、メッセージに属性を追加することもできます。詳細については、「メッセージメタデータ」を参照してください。

Amazon SQS メッセージの受信

この例では、 を使用して Amazon SQS キューからメッセージ AWS SDK for .NET を受信する方法を示します。このキューは、プログラムまたは Amazon SQS コンソールを使用して作成できます。アプリケーションはキューから 1 つのメッセージを読み取り、メッセージを処理 (この例ではコンソールにメッセージ本文を表示) した後で、キューからメッセージを削除します。アプリケーションは、ユーザーがキーボードでキーを入力するまで、これらの手順を繰り返します。

この例と、前のメッセージの送信に関する例を一緒に使用して、Amazon SQS のメッセージフローを確認できます。

以下のセクションでは、この例のスニペットを確認できます。その下には、この例のコードの全文が示されており、そのままビルドして実行できます。

トピック
- メッセージの受信
- メッセージの削除
- コード全文
- その他の考慮事項

メッセージの受信

次のスニペットでは、指定されたキュー URL で特定されるキューからメッセージを受信します。

このトピックの最後で、スニペットが実際に使用されている例を確認できます。

```
//
// 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.)
    });
  }
```

メッセージの削除

次のスニペットでは、指定されたキュー URL で特定されるキューからメッセージを削除します。

このトピックの最後で、スニペットが実際に使用されている例を確認できます。

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

コード全文

このセクションでは、例に関連する参考資料とコードの全文を示します。

SDK リファレンス

NuGet パッケージ:

- AWSSDK.SQS

プログラミング要素:

- 名前空間 Amazon.SQS

  クラス AmazonSQSClient
- 名前空間 Amazon.SQS.Model

  クラス ReceiveMessageRequest

  クラス ReceiveMessageResponse

コード

```
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);
    }
  }
}
```

その他の考慮事項

- ロングポーリングを指定するために、この例では ReceiveMessageAsync メソッドへの各呼び出しで WaitTimeSeconds プロパティを使用しています。

  キューの作成時または更新時に ReceiveMessageWaitTimeSeconds 属性を使用して、キューのすべてのメッセージにロングポーリングを指定することもできます。

  ショートポーリングとロングポーリングの違いに関する詳細については、Amazon Simple Queue Service デベロッパーガイドの「ショートポーリングとロングポーリング」を参照してください。

- メッセージの処理中に受信ハンドルを使用して、メッセージの可視性タイムアウトを変更できます。その方法については、AmazonSQSClient クラスの ChangeMessageVisibilityAsync メソッドを参照してください。

- DeleteMessageAsync メソッドを無条件で呼び出すと、可視性タイムアウトの設定にかかわらず、メッセージがキューから削除されます。

コンピューティングサービス AWS Lambda での の使用

は AWS SDK for .NET をサポートしているため AWS Lambda、サーバーのプロビジョニングや管理を行わずにコードを実行できます。詳細については、AWS Lambda 製品ページとAWS Lambda デベロッパーガイド、特に C# での作業に関するセクションを参照してください。

API

AWS SDK for .NET には APIs が用意されています AWS Lambda。API を使用すると、関数、トリガー、イベントなどの Lambda 機能を操作できます。API の全セットを確認するには、「AWS SDK for .NET API リファレンス」の「Lambda」を参照してください。

Lambda API は NuGetパッケージによって提供されます。

前提条件

開始する前に、環境をセットアップし、プロジェクトを設定したことを確認してください。また、「SDK の使用」の情報を確認してください。

追加情報

.NET Aspire AWS Lambda を使用した の開発については、.NET Aspire AWS との統合「」を参照してください。

トピック

トピック

- 注釈を使用して AWS Lambda 関数を記述する

注釈を使用して AWS Lambda 関数を記述する

Lambda 関数を作成する場合、大量のハンドラーコードの記述や AWS CloudFormation テンプレートの更新などのタスクが必要になることがあります。Lambda アノテーションは、.NET 6 Lambda 関数にかかるこうした負担を軽減するためのフレームワークです。これにより、C# で Lambda をより自然に記述できるようになります。

Lambda Annotations フレームワークを使用する利点の例として、2 つの数値を加算する以下のコードスニペットを考えてみましょう。

Lambda Annotations なし

```
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" } }
        };
    }
}
```

Lambda Annotations あり

```
public class Functions
{
    [LambdaFunction]
    [RestApi("/plus/{x}/{y}")]
    public int Plus(int x, int y)
    {
        return x + y;
    }
}
```

例に示されているように、Lambda Annotations を使用すると特定のボイラープレートコードが不要になります。

フレームワークの使用方法の詳細については、次のリソースを参照してください。

- Lambda アノテーションのAPIと属性に関するドキュメントについては、GitHub のREADMEを参照してください。
- Lambda Annotations のブログ投稿。
- `Amazon.Lambda.Annotations` NuGet パッケージ。
- GitHubの 写真アセット管理プロジェクト。具体的には、PamapiAnnotations フォルダーとプロジェクト README の Lambda アノテーションへのリファレンスを参照してください。

> **Note**
>
> 前述の例は、 の V3 に固有のものです AWS SDK for .NET。SDK の V4 (最新バージョン) で例を使用する場合、 の情報に従って調整する必要がある場合があります[バージョン 4 への移行]。

の高レベルライブラリとフレームワーク AWS SDK for .NET

以下のセクションには、SDK のコア機能には含まれない高レベルライブラリとフレームワークに関する情報が含まれています。これらのライブラリとフレームワークは、SDK のコア機能を使用して特定のタスクを容易にする機能を作成します。

を初めて使用する場合は AWS SDK for .NET、まず[シンプルなアプリケーションの作成]トピックを確認してください。SDK についてわかりやすく説明しています。

開始する前に、[環境をセットアップ]し、[プロジェクトを設定した]ことを確認してください。また、「[SDK の使用]」の情報を確認してください。

トピック

- [AWS .NET のメッセージ処理フレームワーク]
- [での .NET Aspire AWS との統合 AWS SDK for .NET]

AWS .NET のメッセージ処理フレームワーク

AWS Message Processing Framework for .NET は、Amazon Simple Queue Service (SQS)、Amazon Simple Notification Service (SNS)、Amazon EventBridge などの AWS サービスを使用する .NET メッセージ処理アプリケーションの開発を簡素化する AWSネイティブフレームワークです。このフレームワークにより、デベロッパーが記述する必要のある定型コードの量が減り、メッセージを公開または使用する際にビジネスロジックに集中できるようになります。フレームワークが開発を簡素化する方法の詳細については、ブログ記事「[Introducing the AWS Message Processing Framework for .NET (Preview)]」を参照してください。特に最初の部分では、低レベル API コールの使用とフレームワークの使用の違いを示すデモンストレーションを提供します。

メッセージ処理フレームワークは、次のアクティビティと機能をサポートしています。

- SQS にメッセージを送信し、SNS と EventBridge にイベントを発行します。

- 通常、バックグラウンドサービスで使用される長時間実行されるポーラーを使用して、SQS からのメッセージの受信と処理を行います。これには、他のクライアントがメッセージを処理できないように、メッセージの処理中に可視性タイムアウトを管理することが含まれます。
- AWS Lambda 関数でのメッセージの処理。
- FIFO (first-in-first-out) SQS キューと SNS トピック。
- ログ記録用の OpenTelemetry。

これらのアクティビティと機能の詳細については、ブログ記事の「機能」セクションと以下に記載されているトピックを参照してください。

開始する前に、環境をセットアップし、プロジェクトを設定したことを確認してください。また、「SDK の使用」の情報を確認してください。

追加リソース

- NuGet.org の `AWS.Messaging`パッケージ。
- API リファレンス。
- の GitHub リポジトリの README ファイル https://github.com/aws/aws-dotnet-messaging/
- Microsoft からの .NET 依存関係インジェクション。
- Microsoft の .NET 汎用ホスト。

トピック

- Message AWS Processing Framework for .NET の使用を開始する
- AWS Message Processing Framework for .NET を使用してメッセージを発行する
- Message Processing Framework for .NET で AWS メッセージを使用する
- Message AWS Processing Framework for .NET での FIFO の使用
- AWS Message Processing Framework for .NET のログ記録と Open Telemetry
- AWS Message Processing Framework for .NET をカスタマイズする
- AWS Message Processing Framework for .NET のセキュリティ

Message AWS Processing Framework for .NET の使用を開始する

開始する前に、環境をセットアップし、プロジェクトを設定したことを確認してください。また、「SDK の使用」の情報を確認してください。

このトピックでは、メッセージ処理フレームワークの使用を開始するのに役立つ情報を提供します。前提条件と設定情報に加えて、一般的なシナリオの実装方法を示すチュートリアルが用意されています。

前提条件と設定

- アプリケーションに提供する認証情報には、使用するメッセージングサービスとオペレーションに対する適切なアクセス許可が必要です。詳細については、それぞれのデベロッパーガイドの SQS、SNS、EventBridge のセキュリティトピックを参照してください。また、特定のアクセス許可について説明する GitHub の README ファイルの部分も参照してください。
- Message AWS Processing Framework for .NET を使用するには、`AWS.Messaging`NuGet パッケージをプロジェクトに追加する必要があります。例:

```
dotnet add package AWS.Messaging
```

- このフレームワークは、.NET の依存関係インジェクション (DI) サービスコンテナと統合されています。アプリケーションの起動時に を呼び出しAddAWSMessageBusて DI コンテナに追加することで、フレームワークを設定できます。

```
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");
});
```

チュートリアル

このチュートリアルでは、 AWS Message Processing Framework for .NET の使用方法を示します。2 つのアプリケーションを作成します。API エンドポイントでリクエストを受信したときに Amazon SQS キューにメッセージを送信する ASP.NET Core Minimal API と、これらのメッセージをポーリングして処理する長時間実行されるコンソールアプリケーションです。

- このチュートリアルの手順は .NET CLI を優先しますが、.NET CLI や Microsoft Visual Studio などのクロスプラットフォームツールを使用してこのチュートリアルを実行できます。ツールの

詳細については、「」を参照してください用のツールチェーンのインストールと設定 AWS SDK for .NET。

- このチュートリアルでは、認証情報に [default]プロファイルを使用していることを前提としています。また、Amazon SQS メッセージを送受信するための適切なアクセス許可を持つ短期認証情報が利用可能であることを前提としています。詳細については、AWS SDK for .NET を使用したの認証 AWS「」および SQS のセキュリティトピックを参照してください。

> **Note**
>
> このチュートリアルを実行すると、SQS メッセージングのコストが発生する可能性があります。

ステップ

- SQS キューを作成する
- 公開アプリケーションを作成して実行する
- 処理アプリケーションを作成して実行する
- クリーンアップ

SQS キューを作成する

このチュートリアルでは、メッセージを送受信するために SQS キューが必要です。キューは、AWS CLI または に次のいずれかのコマンドを使用して作成できます AWS Tools for PowerShell。返されるキュー URL を書き留めておき、次のフレームワーク設定で指定できるようにします。

AWS CLI

```
aws sqs create-queue --queue-name DemoQueue
```

AWS Tools for PowerShell

```
New-SQSQueue -QueueName DemoQueue
```

公開アプリケーションを作成して実行する

公開アプリケーションを作成して実行するには、次の手順に従います。

1. コマンドプロントまたはターミナルを開きます。.NET プロジェクトを作成できるオペレーティングシステムフォルダを検索するか作成します。
2. そのフォルダで、次のコマンドを実行して .NET プロジェクトを作成します。

```
dotnet new webapi --name Publisher
```

3. 新しいプロジェクトのフォルダに移動します。 AWS Message Processing Framework for .NET への依存関係を追加します。

```
cd Publisher
dotnet add package AWS.Messaging
```

> **Note**
>
> 認証 AWS IAM アイデンティティセンター に を使用している場合は、必ず AWSSDK.SSOと も追加してくださいAWSSDK.SSOOIDC。

4. のコードを次のコードProgram.csに置き換えます。

```
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. 以下のコマンドを実行してください。これにより、Swagger UI でブラウザウィンドウが開き、API を探索してテストできます。

```
dotnet watch run <queue URL created earlier>
```

6. /greeting エンドポイントを開き、試すを選択します。
7. メッセージに senderName と greetingの値を指定し、実行 を選択します。これにより、SQS メッセージを送信する API が呼び出されます。

処理アプリケーションを作成して実行する

次の手順を使用して、処理アプリケーションを作成して実行します。

1. コマンドプロントまたはターミナルを開きます。.NET プロジェクトを作成できるオペレーティングシステムフォルダを検索するか作成します。
2. そのフォルダで、次のコマンドを実行して .NET プロジェクトを作成します。

```
dotnet new console --name Handler
```

3. 新しいプロジェクトのフォルダに移動します。 AWS Message Processing Framework for .NET への依存関係を追加します。また、 Microsoft.Extensions.Hostingパッケージを追加します。これにより、.NET 汎用ホストを介してフレームワークを設定できます。

```
cd Handler
dotnet add package AWS.Messaging
dotnet add package Microsoft.Extensions.Hosting
```

> **Note**
>
> 認証 AWS IAM アイデンティティセンター に を使用している場合は、必ず AWSSDK.SSOと も追加してくださいAWSSDK.SSOOIDC。

4. のコードを次のコードProgram.csに置き換えます。

```
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. 以下のコマンドを実行してください。これにより、長時間実行されるポーラーが開始されます。

```
dotnet run <queue URL created earlier>
```

起動直後に、アプリケーションはこのチュートリアルの最初の部分で送信されたメッセージを受信し、次のメッセージをログに記録します。

```
Received message {greeting} from {senderName}
```

6. Ctrl+C を押してポーラーを停止します。

クリーンアップ

キュー AWS Tools for PowerShell を削除するには、 AWS CLI または に次のいずれかのコマンドを使用します。

AWS CLI

```
aws sqs delete-queue --queue-url "<queue URL created earlier>"
```

AWS Tools for PowerShell

```
Remove-SQSQueue -QueueUrl "<queue URL created earlier>"
```

AWS Message Processing Framework for .NET を使用してメッセージを発行する

AWS Message Processing Framework for .NET は、1 つ以上のメッセージタイプの発行、1 つ以上のメッセージタイプの処理、または同じアプリケーションでの両方の実行をサポートしています。

次のコードは、異なる AWS サービスに異なるメッセージタイプを発行するアプリケーションの設定を示しています。

```
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");
});
```

起動時にフレームワークを登録したら、汎用 をコードIMessagePublisherに挿入します。PublishAsync メソッドを呼び出して、上記で設定されたメッセージタイプを発行します。汎用パブリッシャーは、タイプに基づいてメッセージをルーティングする送信先を決定します。

次の例では、ASP.NET MVC コントローラーがユーザーからChatMessageメッセージとOrderInfoイベントの両方を受信し、それぞれ Amazon SQS と Amazon SNS に発行します。どちらのメッセージタイプも、上記で設定された汎用パブリッシャーを使用して発行できます。

```
[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();
    }
}
```

メッセージを適切な処理ロジックにルーティングするために、フレームワークはメッセージタイプ識別子と呼ばれるメタデータを使用します。デフォルトでは、これはアセンブリ名を含むメッセージの .NET タイプのフルネームです。メッセージの送信と処理の両方を行う場合、メッセージオブジェクトの定義をプロジェクト間で共有すると、このメカニズムはうまく機能します。ただし、メッセージが別の名前空間で再定義されている場合、または他のフレームワークやプログラミング言語とメッセージを交換する場合は、メッセージタイプ識別子を上書きする必要がある場合があります。

```
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");
    });
});
```

サービス固有のパブリッシャー

上記の例では、汎用 を使用しています。これはIMessagePublisher、設定されたメッセージタイプに基づいてサポートされている任意の AWS サービスに発行できます。このフレームワークは、Amazon SQS Amazon SNS 、Amazon EventBridge のサービス固有のパブリッシャーも提供します。これらの特定のパブリッシャーは、そのサービスにのみ適用され、タイプ ISQSPublisher、、ISNSPublisherおよび を使用して挿入できるオプションを公開しますIEventBridgePublisher。

たとえば、SQS FIFO キューにメッセージを送信する場合は、適切なメッセージグループ ID を設定する必要があります。次のコードは、もう一度ChatMessage例を示していますが、現在は ISQSPublisher を使用して SQS 固有のオプションを設定しています。

```
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();
    }
}
```

SNS と EventBridge では、IEventBridgePublisherそれぞれ ISNSPublisherと を使用して同じ操作を実行できます。

```
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>
});
```

デフォルトでは、特定のタイプのメッセージが、事前に設定された送信先に送信されます。ただし、メッセージ固有のパブリッシャーを使用して、1 つのメッセージの送信先を上書きできます。また、メッセージの発行に使用される基盤となる AWS SDK for .NET クライアントを上書きすることもできます。これは、送信先に応じてロールまたは認証情報を変更する必要があるマルチテナントアプリケーションで役立ちます。

```
await _sqsPublisher.SendAsync(message, new SQSOptions
{
    OverrideClient = <override IAmazonSQS client>,
    QueueUrl = <override queue URL>
});
```

Message Processing Framework for .NET で AWS メッセージを使用する

AWS Message Processing Framework for .NET では、フレームワークまたはいずれかのメッセージングサービスを使用して発行されたメッセージを消費できます。メッセージはさまざまな方法で使用できます。その一部を以下に示します。

メッセージハンドラー

メッセージを使用するには、処理するメッセージタイプごとに IMessageHandlerインターフェイスを使用してメッセージハンドラーを実装します。メッセージタイプとメッセージハンドラー間のマッピングは、プロジェクトの起動時に設定されます。

```
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();
```

次のコードは、メッセージのサンプルChatMessageメッセージハンドラーを示しています。

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

外部には、フレームワークで使用されるメタデータMessageEnvelopeが含まれています。そのmessageプロパティはメッセージタイプです (この場合は ChatMessage) 。

MessageProcessStatus.Success( ) に戻って、メッセージが正常に処理され、フレームワークが Amazon SQS キューからメッセージを削除することを示すことができます。を返すとMessageProcessStatus.Failed( )、メッセージはキューに残り、再度処理したり、設定されている場合はデッドレターキューに移動したりできます。

長時間実行されるプロセスでのメッセージの処理

SQS キュー URL AddSQSPollerを使用して を呼び出し、キューBackgroundServiceを継続的にポーリングしてメッセージを処理する長時間実行を開始できます。

```
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();
```

SQS Message Poller の設定

SQS メッセージポーラーは、 を呼び出すSQSMessagePollerOptionsときに によって設定できますAddSQSPoller。

- MaxNumberOfConcurrentMessages - キューから同時に処理するメッセージの最大数。デフォルト値は 10 です。
- WaitTimeSeconds - SQS ReceiveMessage 呼び出しがメッセージがキューに到着するまで待機してから戻る期間 (秒単位) 。メッセージが使用可能な場合、呼び出しは よりも早く返されますWaitTimeSeconds。デフォルト値は 20 です。

メッセージ可視性タイムアウト処理

SQS メッセージには可視性タイムアウト期間があります。あるコンシューマーが特定のメッセージの処理を開始すると、そのコンシューマーはキューに残りますが、複数回処理されないように他のコンシューマーからは非表示になります。再度表示される前にメッセージが処理および削除されない場合、別のコンシューマーが同じメッセージの処理を試みる可能性があります。

フレームワークは、現在処理中のメッセージの可視性タイムアウトを追跡して延長しようとします。この動作は、 を呼び出すSQSMessagePollerOptionsときに で設定できますAddSQSPoller。

- VisibilityTimeout - メッセージを受信した秒単位の時間は、後続の取得リクエストから非表示になります。デフォルト値は 30 です。

- `VisibilityTimeoutExtensionThreshold` - メッセージの可視性タイムアウトが期限切れのこの数秒以内になると、フレームワークは可視性タイムアウトを (別の`VisibilityTimeout`秒だけ) 延長します。デフォルト値は 5 です。
- `VisibilityTimeoutExtensionHeartbeatInterval` - フレームワークが有効期限が切れてから数秒以内にメッセージをチェックし、可視性タイムアウトを延長する`VisibilityTimeoutExtensionThreshold`秒単位の頻度。デフォルト値は 1 です。

次の例では、フレームワークは 1 秒ごとに、まだ処理中のメッセージをチェックします。これらのメッセージが再び表示されてから 5 秒以内の場合、フレームワークは各メッセージの可視性タイムアウトをさらに 30 秒自動的に延長します。

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

AWS Lambda 関数でのメッセージの処理

Message AWS Processing Framework for .NET は、SQS と Lambda の統合で使用できます。これは`AWS.Messaging.Lambda`パッケージによって提供されます。開始するには、README を参照してください。

Message AWS Processing Framework for .NET での FIFO の使用

メッセージの順序付けとメッセージの重複排除が重要なユースケースでは、 AWS Message Processing Framework for .NET はfirst-in-first-out (FIFO) Amazon SQS キューと Amazon SNS トピックをサポートしています。

公開

FIFO キューまたはトピックにメッセージを発行するときは、メッセージが属するグループを指定するメッセージグループ ID を設定する必要があります。グループ内のメッセージは順番に処理されます。これは、SQS 固有および SNS 固有のメッセージパブリッシャーで設定できます。

```
await _sqsPublisher.PublishAsync(message, new SQSOptions
```

```
{
    MessageDeduplicationId = <message-deduplication-id>,
    MessageGroupId = <message-group-id>
});
```

登録中

FIFO キューからのメッセージを処理する場合、フレームワークは、ReceiveMessages呼び出しごとに受信された順序で特定のメッセージグループ内のメッセージを処理します。フレームワークは、で終わるキューで設定すると、このオペレーションモードに自動的に入ります.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();
```

AWS Message Processing Framework for .NET のログ記録と Open Telemetry

AWS Message Processing Framework for .NET はOpenTelemetry がフレームワークによって発行または処理される各メッセージのトレースをログに記録するように計測されます。これは AWS.Messaging.Telemetry.OpenTelemetryパッケージによって提供されます。開始するには、README を参照してください。

> **Note**
>
> ログ記録に関するセキュリティ情報については、「」を参照してくださいAWS Message Processing Framework for .NET のセキュリティ。

AWS Message Processing Framework for .NET をカスタマイズする

AWS Message Processing Framework for .NET は、3 つの異なる「レイヤー」でメッセージを構築、送信、処理します。

1. 最外部レイヤーでは、フレームワークはサービスに固有の AWSネイティブリクエストまたはレスポンスを構築します。例えば、Amazon SQS では、SendMessageリクエストを構築し、サービスによって定義されたMessageオブジェクトと連携します。
2. SQS リクエストとレスポンス内で、フレームワークは MessageBody要素 (Amazon SNS Messageの場合は 、Amazon EventBridge Detailの場合は) を JSON 形式の CloudEvent に設定します。 Amazon SNS これには、メッセージを処理するときに MessageEnvelope オブジェクトでアクセスできるフレームワークによって設定されたメタデータが含まれます。
3. 最も内側のレイヤーでは、CloudEvent JSON オブジェクト内の data 属性に、メッセージとして送受信された .NET オブジェクトの JSON シリアル化が含まれています。

```
{
    "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>"
}
```

メッセージエンベロープの設定方法と読み取り方法をカスタマイズできます。

- "id" はメッセージを一意に識別します。デフォルトでは新しい GUID に設定されていますが、独自の を実装IMessageIdGeneratorして DI コンテナに挿入することで上書きできます。
- "type" は、メッセージをハンドラーにルーティングする方法を制御します。デフォルトでは、メッセージに対応する .NET タイプのフルネームが使用されます。メッセージタイプを AddSQSPublisher、、または を介して送信先にマッピングするときにAddSNSPublisher、 messageTypeIdentifierパラメータを使用してこれを上書きできますAddEventBridgePublisher。
- "source" は、メッセージを送信したシステムまたはサーバーを示します。
  - これは、発行元の関数名 AWS Lambda、Amazon ECS の場合はクラスター名とタスク ARN、Amazon EC2 の場合はインスタンス ID、それ以外の場合はフォールバック値 になります/aws/messaging。

  - これは、 AddMessageSourceまたは AddMessageSourceSuffixで上書きできますMessageBusBuilder。
- "time" を UTC の現在の DateTime に設定します。これは、独自の を実装IDateTimeHandlerし、それを DI コンテナに挿入することで上書きできます。
- "data" には、メッセージとして送受信された .NET オブジェクトの JSON 表現が含まれています。
  - ConfigureSerializationOptions on MessageBusBuilder では、メッセージのシリアル化および逆シリアル化時にSystem.Text.Json.JsonSerializerOptions使用する を設定できます。
  - フレームワークが構築したら、追加の属性を挿入したり、メッセージエンベロープを変換したりするには、 AddSerializationCallbackで を介してそれを実装ISerializationCallbackして登録できますMessageBusBuilder。

AWS Message Processing Framework for .NET のセキュリティ

AWS Message Processing Framework for .NET は、 との通信に に依存し AWS SDK for .NET ています AWS。のセキュリティの詳細については AWS SDK for .NET、「」を参照してくださいこの AWS 製品またはサービスのセキュリティ。

セキュリティ上の理由から、フレームワークはユーザーから送信されたデータメッセージを口グに記録しません。デバッグ目的でこの機能を有効にする場合は、メッセージバスEnableDataMessageLogging()で次のように を呼び出す必要があります。

```
builder.Services.AddAWSMessageBus(bus =>
{
    builder.EnableDataMessageLogging();
});
```

潜在的なセキュリティ問題を発見した場合は、 情報の報告に関するセキュリティポリシーを参照してください。

での .NET Aspire AWS との統合 AWS SDK for .NET

.NET Aspire は、クラウド対応アプリケーションを構築する新しい方法です。特に、分散アプリケーションのコンポーネントを実行、接続、デバッグするローカル環境のオーケストレーションを提供します。クラウド対応アプリケーションの内部開発ループを改善するために、.NET アプリケー

ションを AWS リソースに接続するための .NET Aspire との統合が作成されました。これらの統合は、Aspire.Hosting.AWS NuGet パッケージを通じて利用できます。

次の .NET Aspire 統合を使用できます。

- を通じて AWS リソースをプロビジョニングする機能CloudFormation。この統合は、.NET Aspire AppHost プロジェクト内で使用されます。

  詳細については、ブログ記事「Integrating AWS with .NET Aspire」を参照してください。
- のインストール、設定、Amazon DynamoDB Local AWS SDK for .NET への接続。この統合は、.NET Aspire AppHost プロジェクト内で使用されます。

  詳細については、ブログ記事「Integrating AWS with .NET Aspire」を参照してください。
- AWS Lambda 関数のローカル開発環境を有効にします。この統合は、.NET Aspire AppHost プロジェクト内で使用されます。

  詳細については、ブログ記事「Building and Debugging .NET Lambda applications with .NET ASPIRE (パート 1)」および「Building and Debugging .NET Lambda applications with .NET ASPILE (パート 2)」を参照してください。

  > **Note**
  >
  > これはプレビューリリースの機能に関するプレリリースドキュメントです。このドキュメントは変更される可能性があります。

  この機能はプレビュー版であるため、プレビュー機能をオプトインする必要があります。このプレビュー機能とオプトイン方法の詳細については、GitHub の開発トラッカーの問題を参照してください。

追加情報

Aspire.Hosting.AWS で利用可能な統合の使用方法に関する追加情報と詳細については、以下のリソースを参照してください。

- ブログ記事「Integrating AWS with .NET Aspire」。
- ブログ記事「Building and Debugging .NET Lambda applications with .NET ASPILE (パート 1)」および「Building and Debugging .NET Lambda applications with .NET ASPILE (パート 2)」。

- GitHub の integrations-on-dotnet-aspire-for-aws リポジトリ。
- Aspire.Hosting.AWS NuGet パッケージの詳細な README。

他の AWS サービスと設定のサポート

は、前のセクションで説明したサービスに加えて AWS サービス AWS SDK for .NET をサポートします。サポートされているすべてのサービスの API については、AWS SDK for .NET API リファレンスを参照してください。

は、個々の AWS サービスの名前空間に加えて、次の APIs AWS SDK for .NET も提供します。

[面積]	説明	リソース
AWS サポート	AWS サポートケースと Trusted Advisor 機能へのプログラムによるアクセス。	「Amazon.AWSSupport」および「Amazon.AWSSupport.Model」を参照してください。
全般	ヘルパークラスおよび列挙。	「Amazon」および「Amazon.Util」を参照してください。

SDK for .NET (v4) コード例

このトピックのコード例は、 で AWS SDK for .NET (v4) を使用する方法を示しています AWS。

基本 は、重要なオペレーションをサービス内で実行する方法を示すコード例です。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

シナリオは、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

一部のサービスには、サービス固有のライブラリや関数の活用方法を示す追加のカテゴリ例が含まれています。

サービス

- SDK for .NET (v4) を使用した Aurora の例
- SDK for .NET (v4) を使用した Auto Scaling の例
- SDK for .NET (v4) を使用した Amazon Bedrock の例
- SDK for .NET (v4) を使用した Amazon Bedrock ランタイムの例
- CloudFormation SDK for .NET (v4) を使用した例
- SDK for .NET (v4) を使用した CloudWatch の例
- SDK for .NET (v4) を使用した CloudWatch Logs の例
- SDK for .NET (v4) を使用した Amazon Cognito ID プロバイダーの例
- AWS Control Tower SDK for .NET (v4) を使用した例
- SDK for .NET (v4) を使用した DynamoDB の例
- SDK for .NET (v4) を使用した Amazon EC2 の例
- SDK for .NET (v4) を使用した Amazon ECS の例
- AWS IoT SDK for .NET (v4) を使用した例
- AWS IoT data SDK for .NET (v4) を使用した例
- SDK for .NET (v4) を使用した Amazon Redshift の例
- SDK for .NET (v4) を使用した Amazon S3 の例

SDK for .NET (v4) を使用した Aurora の例

次のコード例は、Aurora で AWS SDK for .NET (v4) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

基本 は、重要なオペレーションをサービス内で実行する方法を示すコード例です。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には完全なソースコードへのリンクが含まれており、コンテキスト内でコードを設定および実行する方法の手順を確認できます。

トピック

- はじめに
- 基本
- アクション

はじめに

Hello Aurora

次のコード例は、Aurora の使用を開始する方法を示しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

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

- API の詳細については、AWS SDK for .NET API リファレンスの「DescribeDBClusters」を参照してください。

基本

基本を学ぶ

次のコード例は、以下の操作方法を示しています。

- カスタム Aurora DB クラスターパラメータグループを作成し、パラメータ値を設定します。
- パラメータグループを使用する DB クラスターを作成する
- データベースを含む DB インスタンスを作成します。
- DB クラスターのスナップショットを作成して、リソースをクリーンアップします。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

コマンドプロンプトからインタラクティブなシナリオを実行します。

```
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;
    }
```

Aurora アクションを管理するためにシナリオによって呼び出されるラッパーメソッド。

```
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;
    }
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の以下のトピックを参照してください。
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - CreateDBClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster
  - DeleteDBClusterParameterGroup
  - DeleteDBInstance
  - DescribeDBClusterParameterGroups
  - DescribeDBClusterParameters
  - DescribeDBClusterSnapshots
  - DescribeDBClusters
  - DescribeDBEngineVersions
  - DescribeDBInstances
  - DescribeOrderableDBInstanceOptions
  - ModifyDBClusterParameterGroup

アクション

`CreateDBCluster`

次の例は、`CreateDBCluster` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
/// <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;
}
```

- API の詳細については、AWS SDK for .NET API リファレンスの「CreateDBCluster」を参照してください。

CreateDBClusterParameterGroup

次の例は、CreateDBClusterParameterGroup を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「CreateDBClusterParameterGroup」を参照してください。

`CreateDBClusterSnapshot`

次の例は、`CreateDBClusterSnapshot` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「CreateDBClusterSnapshot」を参照してください。

CreateDBInstance

次の例は、CreateDBInstance を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
```

```
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「CreateDBInstance」を参照してください。

`DeleteDBCluster`

次の例は、`DeleteDBCluster` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DeleteDBCluster」を参照してください。

DeleteDBClusterParameterGroup

次の例は、DeleteDBClusterParameterGroup を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
/// <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;
}
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DeleteDBClusterParameterGroup」を参照してください。

DeleteDBInstance

次の例は、DeleteDBInstance を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DeleteDBInstance」を参照してください。

`DescribeDBClusterParameterGroups`

次の例は、`DescribeDBClusterParameterGroups` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

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

- API の詳細については、AWS SDK for .NET API リファレンスの「DescribeDBClusterParameterGroups」を参照してください。

`DescribeDBClusterParameters`

次の例は、`DescribeDBClusterParameters` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DescribeDBClusterParameters」を参照してください。

DescribeDBClusterSnapshots

次の例は、DescribeDBClusterSnapshots を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DescribeDBClusterSnapshots」を参照してください。

DescribeDBClusters

次の例は、DescribeDBClusters を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DescribeDBClusters」を参照してください。

`DescribeDBEngineVersions`

次の例は、`DescribeDBEngineVersions` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DescribeDBEngineVersions」を参照してください。

`DescribeDBInstances`

次の例は、`DescribeDBInstances` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   /// <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;
   }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DescribeDBInstances」を参照してください。

DescribeOrderableDBInstanceOptions

次の例は、DescribeOrderableDBInstanceOptions を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   /// <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;
   }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DescribeOrderableDBInstanceOptions」を参照してください。

ModifyDBClusterParameterGroup

次の例は、ModifyDBClusterParameterGroup を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   /// <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;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「ModifyDBClusterParameterGroup」を参照してください。

SDK for .NET (v4) を使用した Auto Scaling の例

次のコード例は、Auto Scaling で AWS SDK for .NET (v4) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

基本 は、重要なオペレーションをサービス内で実行する方法を示すコード例です。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には完全なソースコードへのリンクが含まれており、コンテキスト内でコードを設定および実行する方法の手順を確認できます。

トピック

- はじめに
- 基本
- アクション

はじめに

こんにちは、Auto Scaling

次のコード例は、Auto Scaling の使用を開始する方法を示しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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
        });
```

```
    }
}
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DescribeAutoScalingGroups」を参照してください。

基本

基本を学ぶ

次のコード例は、以下の操作方法を示しています。

- 起動テンプレートとアベイラビリティーゾーンを使用して、Amazon EC2 Auto Scaling グループを作成し、実行中のインスタンスに関する情報を取得します。
- Amazon CloudWatch メトリクスの収集を有効にします。
- グループの希望するキャパシティを更新し、インスタンスが起動するのを待ちます。
- グループ内の最も古いインスタンスを削除します。
- ユーザーのリクエストやキャパシティの変更に応じて発生するスケーリングアクティビティを一覧表示します。
- CloudWatch メトリクスの統計を取得して、リソースをクリーンアップします。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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();
    }
}
```

起動テンプレートとメトリクスを管理するためにシナリオによって呼び出される関数を定義します。これらの関数は、Auto Scaling、Amazon EC2、および CloudWatch アクションをラップします。

```
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;
    }

}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の以下のトピックを参照してください。
  - CreateAutoScalingGroup
  - DeleteAutoScalingGroup
  - DescribeAutoScalingGroups
  - DescribeAutoScalingInstances
  - DescribeScalingActivities
  - DisableMetricsCollection
  - EnableMetricsCollection
  - SetDesiredCapacity
  - TerminateInstanceInAutoScalingGroup
  - UpdateAutoScalingGroup

アクション

`CreateAutoScalingGroup`

次の例は、`CreateAutoScalingGroup` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
/// <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;
        }
    }
```

- 詳細については、AWS SDK for .NET API リファレンスの「CreateAutoScalingGroup」を参照してください。

`DescribeAutoScalingGroups`

次の例は、`DescribeAutoScalingGroups` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DescribeAutoScalingGroups」を参照してください。

DescribeAutoScalingInstances

次の例は、DescribeAutoScalingInstances を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DescribeAutoScalingInstances」を参照してください。

`DescribeScalingActivities`

次の例は、`DescribeScalingActivities` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DescribeScalingActivities」を参照してください。

`DisableMetricsCollection`

次の例は、`DisableMetricsCollection` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DisableMetricsCollection」を参照してください。

`EnableMetricsCollection`

次の例は、`EnableMetricsCollection` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「EnableMetricsCollection」を参照してください。

SetDesiredCapacity

次の例は、SetDesiredCapacity を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「SetDesiredCapacity」を参照してください。

`TerminateInstanceInAutoScalingGroup`

次の例は、`TerminateInstanceInAutoScalingGroup` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「TerminateInstanceInAutoScalingGroup」を参照してください。

`UpdateAutoScalingGroup`

次の例は、`UpdateAutoScalingGroup` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「UpdateAutoScalingGroup」を参照してください。

SDK for .NET (v4) を使用した Amazon Bedrock の例

次のコード例は、Amazon Bedrock で AWS SDK for .NET (v4) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には完全なソースコードへのリンクが含まれており、コンテキスト内でコードを設定および実行する方法の手順を確認できます。

トピック

- はじめに
- アクション

はじめに

Hello Amazon Bedrock

次のコード例は、Amazon Bedrock の使用を開始する方法を示しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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)}");
    }
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「ListFoundationModels」を参照してください。

アクション

`ListFoundationModels`

次の例は、`ListFoundationModels` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

利用可能な Bedrock 基盤モデルを一覧表示します。

```
    /// <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);
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「ListFoundationModels」を参照してください。

SDK for .NET (v4) を使用した Amazon Bedrock ランタイムの例

次のコード例は、Amazon Bedrock ランタイムで AWS SDK for .NET (v4) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

各例には完全なソースコードへのリンクが含まれており、コンテキスト内でコードを設定および実行する方法の手順を確認できます。

トピック

- Anthropic Claude
- Cohere Command
- Meta Llama
- Mistral AI

Anthropic Claude

Converse

次のコード例は、Bedrock の Converse API を使用して Anthropic Claude にテキストメッセージを送信する方法を示しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

Bedrock の Converse API を使用して、Anthropic Claude にテキストメッセージを送信します。

```
// 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;
}
```

- API の詳細については、AWS SDK for .NET API リファレンスの「Converse」を参照してください。

ConverseStream

次のコード例は、Bedrock の Converse API を使用して Anthropic Claude にテキストメッセージを送信し、レスポンスストリームをリアルタイムで処理する方法を示しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

Bedrock の Converse API を使用して Anthropic Claude にテキストメッセージを送信し、レスポンスストリームをリアルタイムで処理します。

```
// 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;
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「ConverseStream」を参照してください。

InvokeModel

次のコード例は、Invoke Model API を使用して Anthropic Claude にテキストメッセージを送信する方法を示しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

Invoke Model API を使用してテキストメッセージを送信します。

```
// 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;
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「InvokeModel」を参照してください。

Cohere Command

Converse

次のコード例は、Bedrock の Converse API を使用して Cohere Command にテキストメッセージを送信する方法を示しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

Bedrock の Converse API を使用して Cohere Command にテキストメッセージを送信します。

```
// 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;
}
```

- API の詳細については、AWS SDK for .NET API リファレンスの「Converse」を参照してください。

ConverseStream

次のコード例は、Bedrock の Converse API を使用して Cohere Command にテキストメッセージを送信し、レスポンスストリームをリアルタイムで処理する方法を示しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

Bedrock の Converse API を使用して Cohere Command にテキストメッセージを送信し、レスポンスストリームをリアルタイムで処理します。

```
// 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;
}
```

- API の詳細については、AWS SDK for .NET API リファレンスの「ConverseStream」を参照してください。

Meta Llama

Converse

次のコード例は、Bedrock の Converse API を使用して Meta Llama にテキストメッセージを送信する方法を示しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

Bedrock の Converse API を使用して Meta Llama にテキストメッセージを送信します。

```
// 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;
}
```

- API の詳細については、AWS SDK for .NET API リファレンスの「Converse」を参照してください。

ConverseStream

次のコード例は、Bedrock の Converse API を使用して Meta Llama にテキストメッセージを送信し、レスポンスストリームをリアルタイムで処理する方法を示しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

Bedrock の Converse API を使用して Meta Llama にテキストメッセージを送信し、レスポンスストリームをリアルタイムで処理します。

```
// 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;
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「ConverseStream」を参照してください。

Mistral AI

Converse

次のコード例は、Bedrock の Converse API を使用して Mistral にテキストメッセージを送信する方法を示しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

Bedrock の Converse API を使用して Mistral にテキストメッセージを送信します。

```
// 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;
}
```

- API の詳細については、AWS SDK for .NET API リファレンスの「Converse」を参照してください。

ConverseStream

次のコード例は、Bedrock の Converse API を使用して Mistral にテキストメッセージを送信し、レスポンスストリームをリアルタイムで処理する方法を示しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

Bedrock の Converse API を使用して Mistral にテキストメッセージを送信し、レスポンスストリームをリアルタイムで処理します。

```
// 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;
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「ConverseStream」を参照してください。

CloudFormation SDK for .NET (v4) を使用した例

次のコード例は、 で AWS SDK for .NET (v4) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています CloudFormation。

各例には完全なソースコードへのリンクが含まれており、コンテキスト内でコードを設定および実行する方法の手順を確認できます。

トピック

- はじめに

はじめに

こんにち CloudFormationは

次のコード例は、 CloudFormationの使用を開始する方法を示しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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;
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DescribeStackResources」を参照してください。

SDK for .NET (v4) を使用した CloudWatch の例

次のコード例は、CloudWatch で AWS SDK for .NET (v4) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

基本 は、重要なオペレーションをサービス内で実行する方法を示すコード例です。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には完全なソースコードへのリンクが含まれており、コンテキスト内でコードを設定および実行する方法の手順を確認できます。

トピック

- はじめに
- 基本
- アクション

はじめに

Hello CloudWatch

次のコード例は、CloudWatch の使用を開始する方法を示しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

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

- API の詳細については、AWS SDK for .NET API リファレンスの「ListMetrics」を参照してください。

基本

基本を学ぶ

次のコード例は、以下の操作方法を示しています。

- CloudWatch の名前空間とメトリクスを一覧表示します。
- メトリクスと予想請求額の統計の取得
- ダッシュボードの作成と更新
- メトリクスの作成とデータの追加
- アラームの作成/トリガーとアラーム履歴の表示
- 異常ディテクターの追加
- メトリクス画像を取得し、リソースをクリーンアップします。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

コマンドプロンプトからインタラクティブなシナリオを実行します。

```
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;
    }
}
```

CloudWatch アクションのシナリオで使用されるラッパーメソッドです。

```
/// <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;
    }
}
```

シナリオの settings.json 値の例。

```
{
  "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
```

```
        }
      }
    ]
  }
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の以下のトピックを参照してください。
  - DeleteAlarms
  - DeleteAnomalyDetector
  - DeleteDashboards
  - DescribeAlarmHistory
  - DescribeAlarms
  - DescribeAlarmsForMetric
  - DescribeAnomalyDetectors
  - GetMetricData
  - GetMetricStatistics
  - GetMetricWidgetImage
  - ListMetrics
  - PutAnomalyDetector
  - PutDashboard
  - PutMetricAlarm
  - PutMetricData

アクション

`DeleteAlarms`

次の例は、`DeleteAlarms` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
/// <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;
}
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DeleteAlarms」を参照してください。

`DeleteAnomalyDetector`

次の例は、`DeleteAnomalyDetector` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DeleteAnomalyDetector」を参照してください。

`DeleteDashboards`

次の例は、`DeleteDashboards` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DeleteDashboards」を参照してください。

`DescribeAlarmHistory`

次の例は、`DescribeAlarmHistory` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DescribeAlarmHistory」を参照してください。

`DescribeAlarms`

次の例は、`DescribeAlarms` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DescribeAlarms」を参照してください。

`DescribeAlarmsForMetric`

次の例は、`DescribeAlarmsForMetric` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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>();
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DescribeAlarmsForMetric」を参照してください。

`DescribeAnomalyDetectors`

次の例は、`DescribeAnomalyDetectors` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DescribeAnomalyDetectors」を参照してください。

`DisableAlarmActions`

次の例は、`DisableAlarmActions` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
    }
```

- API 詳細については、AWS SDK for .NET API リファレンスの「DisableAlarmActions」を参照してください。

EnableAlarmActions

次の例は、EnableAlarmActions を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「EnableAlarmActions」を参照してください。

GetDashboard

次の例は、GetDashboard を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
/// <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;
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「GetDashboard」を参照してください。

`GetMetricData`

次の例は、`GetMetricData` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「GetMetricData」を参照してください。

GetMetricStatistics

次の例は、GetMetricStatistics を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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>();
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「GetMetricStatistics」を参照してください。

GetMetricWidgetImage

次の例は、GetMetricWidgetImage を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「GetMetricWidgetImage」を参照してください。

`ListDashboards`

次の例は、`ListDashboards` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「ListDashboards」を参照してください。

`ListMetrics`

次の例は、`ListMetrics` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「ListMetrics」を参照してください。

PutAnomalyDetector

次の例は、PutAnomalyDetector を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「PutAnomalyDetector」を参照してください。

`PutDashboard`

次の例は、`PutDashboard` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
/// <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>();
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「PutDashboard」を参照してください。

`PutMetricAlarm`

次の例は、`PutMetricAlarm` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「PutMetricAlarm」を参照してください。

PutMetricData

次の例は、PutMetricData を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。完全な例を見つけて、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「PutMetricData」を参照してください。

SDK for .NET (v4) を使用した CloudWatch Logs の例

次のコード例は、CloudWatch Logs で AWS SDK for .NET (v4) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

シナリオは、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック

- アクション
- シナリオ

アクション

`GetQueryResults`

次の例は、`GetQueryResults` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
/// <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;
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「GetQueryResults」を参照してください。

`StartQuery`

次の例は、`StartQuery` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「StartQuery」を参照してください。

シナリオ

大規模なクエリを実行する

次のコード例は、CloudWatch Logs を使用して 10,000 を超えるレコードをクエリする方法を示しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

これは、大規模なクエリシナリオを示すメインワークフローです。

```
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();
    }
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の以下のトピックを参照してください。
  - GetQueryResults
  - StartQuery

SDK for .NET (v4) を使用した Amazon Cognito ID プロバイダーの例

次のコード例は、Amazon Cognito ID プロバイダーで AWS SDK for .NET (v4) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック

- アクション

アクション

`ListUserPools`

次の例は、`ListUserPools` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「ListUserPools」を参照してください。

AWS Control Tower SDK for .NET (v4) を使用した例

次のコード例は、 で AWS SDK for .NET (v4) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています AWS Control Tower。

基本 は、重要なオペレーションをサービス内で実行する方法を示すコード例です。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には完全なソースコードへのリンクが含まれており、コンテキスト内でコードを設定および実行する方法の手順を確認できます。

トピック

- はじめに

- 基本
- アクション

はじめに

こんにち AWS Control Towerは

次のコード例は、 AWS Control Towerの使用を開始する方法を示しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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}");
        }
    }
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「ListBaselines」を参照してください。

基本

基本を学ぶ

次のコード例は、以下の操作方法を示しています。

- ランディングゾーンを一覧表示します。
- ベースラインを一覧表示、有効化、取得、リセット、無効化する。
- コントロールを一覧表示、有効化、取得、無効化する。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

AWS Control Tower 機能を示すインタラクティブなシナリオを実行します。

```
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;
    }
}
```

Aurora アクションを管理するためにシナリオによって呼び出されるラッパーメソッド。

```
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;
        }
    }

}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の以下のトピックを参照してください。
  - CreateLandingZone
  - DeleteLandingZone
  - DisableBaseline
  - DisableControl

- EnableBaseline
- EnableControl
- GetControlOperation
- GetLandingZoneOperation
- ListBaselines
- ListEnabledBaselines
- ListEnabledControls
- ListLandingZones
- ResetEnabledBaseline

アクション

`DisableBaseline`

次の例は、DisableBaseline を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   /// <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;
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DisableBaseline」を参照してください。

DisableControl

次の例は、DisableControl を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   /// <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;
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DisableControl」を参照してください。

`EnableBaseline`

次の例は、`EnableBaseline` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「EnableBaseline」を参照してください。

`EnableControl`

次の例は、`EnableControl` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「EnableControl」を参照してください。

GetBaselineOperation

次の例は、GetBaselineOperation を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「GetBaselineOperation」を参照してください。

`GetControlOperation`

次の例は、`GetControlOperation` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「GetControlOperation」を参照してください。

`ListBaselines`

次の例は、`ListBaselines` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「ListBaselines」を参照してください。

`ListEnabledBaselines`

次の例は、`ListEnabledBaselines` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「ListEnabledBaselines」を参照してください。

ListEnabledControls

次の例は、ListEnabledControls を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「ListEnabledControls」を参照してください。

`ListLandingZones`

次の例は、`ListLandingZones` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「ListLandingZones」を参照してください。

`ResetEnabledBaseline`

次の例は、`ResetEnabledBaseline` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「ResetEnabledBaseline」を参照してください。

SDK for .NET (v4) を使用した DynamoDB の例

次のコード例は、DynamoDB で AWS SDK for .NET (v4) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

基本 は、重要なオペレーションをサービス内で実行する方法を示すコード例です。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には完全なソースコードへのリンクが含まれており、コンテキスト内でコードを設定および実行する方法の手順を確認できます。

トピック

- はじめに
- 基本
- アクション

はじめに

Hello DynamoDB

次のコード例は、DynamoDB の使用を開始する方法を示しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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}");
        }
    }
}
```

- API の詳細については、AWS SDK for .NET API リファレンスの「ListTables」を参照してください。

基本

基本を学ぶ

次のコード例は、以下の操作方法を示しています。

- 映画データを保持できるテーブルを作成します。
- テーブルに 1 つの映画を入れ、取得して更新する。
- サンプル JSON ファイルから映画データをテーブルに書き込む。
- 特定の年にリリースされた映画を照会する。
- 一定期間内に公開された映画をスキャンします。
- テーブルから映画を削除し、テーブルを削除します。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

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

挿入されたクライアントをテーブルオペレーションに使用します。

```
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;
    }
```

ムービーデータを含めるテーブルを作成します。

```
    /// <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;
        }
```

```
    }
```

1 つのムービーをテーブルに追加します。

```
    /// <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;
        }
    }
```

テーブルの 1 つの項目を更新します。

```
    /// <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;
        }
    }
```

ムービーテーブルから 1 つの項目を取得します。

```
   /// <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;
        }
    }
```

項目のバッチをムービーテーブルに書き込みます。

```
    /// <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;
        }
    }
```

テーブルから 1 つの項目を削除します。

```
    /// <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;
        }
    }
```

特定の年にリリースされたムービーのテーブルにクエリを実行します。

```
    /// <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;
        }
    }
```

数年にわたってリリースされたムービーのテーブルをスキャンします。

```
    /// <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;
        }
    }
```

ムービーテーブルを削除します。

```
    /// <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;
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の以下のトピックを参照してください。
  - BatchWriteItem

- CreateTable
- DeleteItem
- DeleteTable
- DescribeTable
- GetItem
- PutItem
- Query
- Scan
- UpdateItem

アクション

`BatchWriteItem`

次の例は、`BatchWriteItem` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

項目のバッチをムービーテーブルに書き込みます。

```
/// <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;
        }
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「BatchWriteItem」を参照してください。

CreateTable

次の例は、CreateTable を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「CreateTable」を参照してください。

`DeleteItem`

次の例は、`DeleteItem` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DeleteItem」を参照してください。

DeleteTable

次の例は、DeleteTable を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DeleteTable」を参照してください。

`GetItem`

次の例は、`GetItem` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「GetItem」を参照してください。

PutItem

次の例は、PutItem を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「PutItem」を参照してください。

Query

次の例は、Query を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「Query」を参照してください。

Scan

次の例は、Scan を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「Scan」を参照してください。

`UpdateItem`

次の例は、`UpdateItem` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「UpdateItem」を参照してください。

SDK for .NET (v4) を使用した Amazon EC2 の例

次のコード例は、Amazon EC2 で AWS SDK for .NET (v4) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

各例には完全なソースコードへのリンクが含まれており、コンテキスト内でコードを設定および実行する方法の手順を確認できます。

トピック

- はじめに

はじめに

Hello Amazon EC2

次のコード例は、Amazon EC2 の使用を開始する方法を示しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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}");
        }
    }
}
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DescribeSecurityGroups」を参照してください。

SDK for .NET (v4) を使用した Amazon ECS の例

次のコード例は、Amazon ECS で AWS SDK for .NET (v4) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

各例には完全なソースコードへのリンクが含まれており、コンテキスト内でコードを設定および実行する方法の手順を確認できます。

トピック

- はじめに

はじめに

Hello Amazon ECS

次のコード例は、Amazon ECS の使用を開始する方法を示しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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.");
        }

    }
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「ListClusters」を参照してください。

AWS IoT SDK for .NET (v4) を使用した例

次のコード例は、 で AWS SDK for .NET (v4) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています AWS IoT。

基本 は、重要なオペレーションをサービス内で実行する方法を示すコード例です。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には完全なソースコードへのリンクが含まれており、コンテキスト内でコードを設定および実行する方法の手順を確認できます。

トピック

- はじめに
- 基本
- アクション

はじめに

こんにち AWS IoTは

次のコード例は、 AWS IoTの使用を開始する方法を示しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
/// <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}");
        }
    }
}
```

- API の詳細については、AWS SDK for .NET API リファレンスの「listThings」を参照してください。

基本

基本を学ぶ

次のコード例は、以下の操作方法を示しています。

- AWS IoT モノを作成します。
- デバイス証明書を生成します。
- 属性を使用して AWS IoT モノを更新します。
- 一意のエンドポイントを返します。
- AWS IoT 証明書を一覧表示します。
- AWS IoT シャドウを更新します。
- 状態情報を書き込みます。
- ルールを作成する。
- ルールを一覧表示します。
- モノの名前を使用してモノを検索します。
- AWS IoT モノを削除します。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

AWS IoT 機能を示すインタラクティブなシナリオを実行します。

```
/// <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;
    }
}
```

AWS IoT SDK メソッドのラッパークラス。

```
/// <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>();
        }
    }

}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の以下のトピックを参照してください。
  - AttachThingPrincipal
  - CreateKeysAndCertificate
  - CreateThing
  - CreateTopicRule
  - DeleteCertificate
  - DeleteThing
  - DeleteTopicRule
  - DescribeEndpoint
  - DescribeThing
  - DetachThingPrincipal
  - ListCertificates
  - ListThings
  - SearchIndex
  - UpdateIndexingConfiguration
  - UpdateThing

アクション

`AttachThingPrincipal`

次の例は、`AttachThingPrincipal` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「AttachThingPrincipal」を参照してください。

CreateKeysAndCertificate

次の例は、CreateKeysAndCertificate を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   /// <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;
       }
   }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「CreateKeysAndCertificate」を参照してください。

`CreateThing`

次の例は、`CreateThing` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「CreateThing」を参照してください。

`CreateTopicRule`

次の例は、`CreateTopicRule` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
            }
        }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「CreateTopicRule」を参照してください。

DeleteCertificate

次の例は、DeleteCertificate を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DeleteCertificate」を参照してください。

`DeleteThing`

次の例は、DeleteThing を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DeleteThing」を参照してください。https://docs.aws.amazon.com/goto/DotNetSDKV4/iot-2015-05-28/DeleteThing

`DescribeEndpoint`

次の例は、`DescribeEndpoint` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DescribeEndpoint」を参照してください。

`DetachThingPrincipal`

次の例は、`DetachThingPrincipal` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DetachThingPrincipal」を参照してください。

ListCertificates

次の例は、ListCertificates を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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>();
        }
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「ListCertificates」を参照してください。

ListThings

次の例は、ListThings を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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>();
        }
    }
```

- API の詳細については、AWS SDK for .NET 「 API リファレンス」のListThings」を参照してください。

`SearchIndex`

次の例は、SearchIndex を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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>();
        }
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「SearchIndex」を参照してください。

`UpdateThing`

次の例は、UpdateThing を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「UpdateThing」を参照してください。

AWS IoT data SDK for .NET (v4) を使用した例

次のコード例は、 で AWS SDK for .NET (v4) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています AWS IoT data。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には完全なソースコードへのリンクが含まれており、コードの設定方法と実行方法に関する手順を確認できます。

トピック

- アクション

アクション

`GetThingShadow`

次の例は、`GetThingShadow` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
    }
```

- API の詳細については、「 API リファレンスAWS SDK for .NET 」の「GetThingShadow」を参照してください。

`UpdateThingShadow`

次の例は、UpdateThingShadow を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   /// <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;
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「UpdateThingShadow」を参照してください。

SDK for .NET (v4) を使用した Amazon Redshift の例

次のコード例は、Amazon Redshift で AWS SDK for .NET (v4) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

基本 は、重要なオペレーションをサービス内で実行する方法を示すコード例です。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

各例には完全なソースコードへのリンクが含まれており、コンテキスト内でコードを設定および実行する方法の手順を確認できます。

トピック

- はじめに
- 基本
- アクション

はじめに

Hello Amazon Redshift

次のコード例は、Amazon Redshift の使用を開始する方法を示しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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}");
        }
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DescribeClusters」を参照してください。

基本

基本を学ぶ

次のコード例は、以下の操作方法を示しています。

- Redshift クラスターを作成します。
- クラスター内のデータベースを一覧表示します。
- Movies という名前のテーブルを作成します。
- Movies テーブルにデータを入力します。
- Movies テーブルに対して年に基づくクエリを実行します。
- Redshift クラスターを変更します。
- Amazon Redshift クラスターを削除します。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

オペレーションを管理する Redshift ラッパークラスを作成します。

```
/// <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}");
    }
}
```

Redshift の基本機能を示すインタラクティブなシナリオを実行します。

```
/// <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; }
    }
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の以下のトピックを参照してください。
  - CreateCluster
  - DescribeClusters
  - DescribeStatement
  - ExecuteStatement
  - GetStatementResult
  - ListDatabasesPaginator
  - ModifyCluster

アクション

`CreateCluster`

次の例は、`CreateCluster` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   /// <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;
        }
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「CreateCluster」を参照してください。

`DeleteCluster`

次の例は、`DeleteCluster` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DeleteCluster」を参照してください。

`DescribeClusters`

次の例は、`DescribeClusters` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
       }
```

```
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DescribeClusters」を参照してください。

`DescribeStatement`

次の例は、`DescribeStatement` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DescribeStatement」を参照してください。

`GetStatementResult`

次の例は、`GetStatementResult` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「GetStatementResult」を参照してください。

`ListDatabases`

次の例は、`ListDatabases` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「ListDatabases」を参照してください。

ModifyCluster

次の例は、ModifyCluster を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
```

```
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「ModifyCluster」を参照してください。

SDK for .NET (v4) を使用した Amazon S3 の例

次のコード例は、Amazon S3 で AWS SDK for .NET (v4) を使用してアクションを実行し、一般的なシナリオを実装する方法を示しています。

基本 は、重要なオペレーションをサービス内で実行する方法を示すコード例です。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

シナリオは、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

各例には完全なソースコードへのリンクが含まれており、コンテキスト内でコードを設定および実行する方法の手順を確認できます。

トピック

- はじめに
- 基本
- アクション
- シナリオ

はじめに

Hello Amazon S3

次のコード例は、Amazon S3 の使用を開始する方法を示しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
/// <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}");
        }
    }
}
```

- API の詳細については、AWS SDK for .NET API リファレンスの「ListBuckets」を参照してください。

基本

基本を学ぶ

次のコード例は、以下の操作方法を示しています。

- バケットを作成し、そこにファイルをアップロードします。
- バケットからオブジェクトをダウンロードします。
- バケット内のサブフォルダにオブジェクトをコピーします。

- バケット内のオブジェクトを一覧表示します。
- バケットオブジェクトとバケットを削除します。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

Amazon S3 機能を示すインタラクティブなシナリオを実行します。

```
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;
        }
    }
}
```

Amazon S3 SDK メソッドのラッパークラス。

```
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;
        }
    }

}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の以下のトピックを参照してください。
  - CopyObject
  - CreateBucket
  - DeleteBucket
  - DeleteObjects
  - GetObject
  - ListObjectsV2
  - PutObject

アクション

`CopyObject`

次の例は、CopyObject を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
/// <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;
    }
```

```
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「CopyObject」を参照してください。

`CreateBucket`

次の例は、`CreateBucket` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
/// <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;
        }
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「CreateBucket」を参照してください。

`CreatePresignedPost`

次の例は、`CreatePresignedPost` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

署名付き POST URL を作成します。

```
    /// <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);
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「CreatePresignedPost」を参照してください。

DeleteBucket

次の例は、DeleteBucket を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
/// <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;
    }
}
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DeleteBucket」を参照してください。

`DeleteObjects`

次の例は、`DeleteObjects` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DeleteObjects」を参照してください。

`GetObject`

次の例は、`GetObject` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
/// <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;
        }
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「GetObject」を参照してください。

`ListObjectsV2`

次の例は、`ListObjectsV2` を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「ListObjectsV2」を参照してください。

PutObject

次の例は、PutObject を使用する方法を説明しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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;
        }
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「PutObject」を参照してください。

シナリオ

署名付き URL を作成する

次のコード例は、Amazon S3 の署名付き URL を作成し、オブジェクトをアップロードする方法を示しています。

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

署名付き POST URL を作成してブラウザからの直接アップロードに使用します。

```
/// <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");
```

```
            }
        }
    }
}
```

のプロジェクトの移行 AWS SDK for .NET

このセクションでは、 のバージョン 4 の移行タスクに関する情報と AWS SDK for .NET、それらのタスクを実行する方法について説明します。以前のバージョンへの移行 (V2 から V3 への移行や、ある V3 サブバージョンから別のバージョンへの移行など) の詳細については、「 AWS SDK for .NET デベロッパーガイド (バージョン 3)」の「プロジェクトを移行する」を参照してください。

「最新情報」および「サポートされているプラットフォーム」も参照してください。

トピック

- のバージョン 4 への移行 AWS SDK for .NET
- .NET Standard 1.3 からの移行

のバージョン 4 への移行 AWS SDK for .NET

AWS SDK for .NET バージョン 4 (V4) には、 SDK のバージョン 3 (V3) からの重大な変更が多数あります。このトピックでは、バージョン 4 の重大な変更と、V3 から環境またはコードを移行するために必要となる可能性のある作業について説明します。SDK のその他の注目すべき変更の詳細については、次のリソースを参照してください。

- GitHub の開発トラッカーの問題: https://github.com/aws/aws-sdk-net/issues/3362。
- ブログ記事「プレビュー 1 of AWS SDK for .NET V4」。
- ブログ記事「プレビュー 4 of AWS SDK for .NET V4」。
- ブログ記事「General Availability of AWS SDK for .NET V4.0」。

特定のランタイムライブラリまたは .NET Framework の最小バージョンが必要です。

.NET Framework 3.5 ターゲットが の V4 から削除されました AWS SDK for .NET。その結果、SDK は .NET Framework 3.5 をサポートしなくなりました。このバージョンの SDK は .NET Framework 4.7.2 に対してコンパイルされ、.NET 4.0 ランタイムで実行されます。詳細については、「サポートされているプラットフォーム」を参照してください。

値のタイプ

リクエストとレスポンスの作成に使用されるクラスで値タイプを使用するプロパティは、null 可能な値タイプを使用するように変更されました。次のタイプのプロパティが変更されました。

- bool が に変更されました bool?
- double が に変更されました double?
- int が に変更されました int?
- float が に変更されました float?
- long が に変更されました long?
- Datetime が に変更されました Datetime?

この変更の詳細については、ブログ記事V4 の AWS SDK for .NET プレビュー 1」を参照してください。

コレクション

リクエストとレスポンスの作成に使用されるクラスでコレクションを使用するプロパティは、デフォルトで になりましたnull。そのため、コードは、コレクションを使用する前に、コレクションが null でないことを確認する必要があります。例えば、次のようになります。

```
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.
    }
}
```

コレクションを初期化する V3 の動作は、 Amazon.AWSConfigs.InitializeCollectionsを に設定することで復元できますtrue。このプロパティは、V4 にアップグレードする前にこの動作の変更を試すユーザーにも V3 に存在します。 V4

この変更の詳細については、ブログ記事V4 の AWS SDK for .NET プレビュー 1」を参照してください。

AWS Security Token Service (STS)

- リージョンエンドポイント

  に依存する認証情報プロバイダーを使用する場合 AWS STS、呼び出しは常にリージョンエンドポイントを使用します。これは、 SDK の V3 とは異なります。SDK は、設定されたus-east-1リージョンに関係なく、パブリックパーティションで実行するときにデフォルトでリージョンを使用していました。

- **StsRegionalEndpointsValue**列挙型

  列挙型が Amazon.Runtime StsRegionalEndpointsValue 名前空間から削除されました。その列挙型を使用するコードはすべて削除する必要があります。

- **STSAssumeRoleAWSCredentials** クラス

  廃止された STS 継承ロール認証情報プロバイダー STSAssumeRoleAWSCredentialsが Amazon.SecurityToken 名前空間から削除されました。代わりに、Amazon.Runtime の AssumeRoleAWSCredentials を使用します。

に関連する変更 **ClientConfig**

Amazon.Runtime.ClientConfig クラスは、AmazonS3Config などのサービスクライアント設定クラスのベースクラスです。このベースクラスには、次の変更が行われました。

- デフォルトの再試行モード

  RetryMode プロパティのデフォルトは Standardではなく ですLegacy。その結果、Legacy値は Amazon.Runtime.RequestRetryMode 列挙型から削除されました。

- デフォルト設定モード

  DefaultConfigurationMode プロパティのデフォルトは Standardではなく ですLegacy。その結果、Legacy値は Amazon.Runtime.DefaultConfigurationMode 列挙型から削除されました。

- **ReadWriteTimeout**プロパティ

  古いプロパティは、.NET Framework 4.7.2 を除くすべてのターゲットから削除ReadWriteTimeoutされました。

AWSSDK.Extensions.NETCore.Setup NuGet パッケージ

AWSSDK.Extensions.NETCore.Setup NuGet パッケージが更新され、SDK の V3 に存在する問題が軽減され、パッケージがネイティブ AOT に対して安全になりました。これらの変更を以下に要約します。詳細については、GitHub の aws-sdk-net リポジトリの PR 3353 を参照してください。

- **`DefaultClientConfig` クラス**

  `DefaultClientConfig` クラスは、サービスクライアント設定ベースクラス Amazon.Runtime.ClientConfig から継承されなくなりました。からの関連プロパティ`ClientConfig`は、null 可能な値型`DefaultClientConfig`を使用して でレプリケートされています。この変更により、サービスクライアント用に作成される設定に値をコピー`DefaultClientConfig`するときに、値がいつに設定されているかを検出できます。

  この変更の特定の結果の 1 つは、 `DefaultClientConfig.HttpClientFactory`が V4 で利用できなくなったことです。代わりに `AWSConfigs.HttpClientFactory` を使用します。詳細については、GitHub 問題 3790 を参照してください。

  この変更のもう 1 つの結果は、サービス設定オブジェクトを受け入れた`IConfiguration.GetAWSOptions`拡張機能メソッドの一般的なオーバーロードが削除されたことです。代わりに非汎用オーバーロードを使用する必要があります。SDK はサービス固有の設定の入力を自動的に処理します。詳細については、GitHub 問題 3866 を参照してください。

- ネイティブ AOT

  C# 11 静的インターフェイスメソッドを使用するサービスクライアントを作成するための新しいメカニズムがパッケージに追加されました。この変更により、アセンブリタイプのロードを実行してサービスクライアントのインスタンスを作成する必要がなくなります。これには、ネイティブ AOT と互換性のないサービスクライアントタイプを計算するためのサービスインターフェイス名の文字列操作が含まれます。この変更は .NET 8 以降でのみ使用できます。古いバージョンでは引き続き元のメカニズムが使用されます。

このパッケージの詳細については、「」を参照してくださいAWSSDK.Extensions.NETCore.Setup および IConfiguration。このパッケージのソースコードは GitHub の https://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.NETCore.Setup にあります。

CookieSigner および UrlSigner

Amazon CloudFront の CookieSignerおよび UrlSigner拡張機能は、AWSSDK.Extensions.CloudFront.Signers という別の拡張機能パッケージに移動されました。この変更は、OpenSSL 3 をサポートし、BouncyCastle.Cryptography に依存するためです。

このパッケージのソースコードは GitHub の https://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.CloudFront.Signers にあります。

DateTime と UTC DateTime

一部の V3 クラスには、「廃止」または「廃止」とマークされた DateTime プロパティと、代替の UTC DateTime プロパティがあります。これらのクラスでは、古い DateTime プロパティが削除され、UTC DateTime プロパティの名前が DateTime プロパティの元の名前に変更されました。

この変更が実装されたクラスの例を以下に示します。

- DescribeSpotPriceHistoryRequest:
  - 古いStartTimeプロパティが削除され、StartTimeUtcプロパティの名前が「StartTime」に変更されました。
  - 古いEndTimeプロパティが削除され、EndTimeUtcプロパティの名前が「EndTime」に変更されました。
- CreateFleetRequest
  - 古いValidFromプロパティが削除され、ValidFromUtcプロパティの名前が「ValidFrom」に変更されました。
  - 古いValidUntilプロパティが削除され、ValidUntilUtcプロパティの名前が「ValidUntil」に変更されました。

この変更により、アプリケーションが元の古い DateTime プロパティを使用している場合、オフセット時間が発生する可能性があります。UTC DateTime プロパティを使用するコードでは、コンパイル時間エラーが発生します。

DateTime 解析

DateTimeUnmarshaller クラスが更新されました。このクラスは、DateTime 文字列をローカル時間として解析して返していました。場合によっては、これらの値は以前の更新のために UTC に変換されていましたが、必ずしも変換されないことがあります。これで、マーシャリングされていない

DateTime 文字列は UTC と見なされ、UTC として指定およびマーシャリング解除されます。この更新には、以下の動作の変更が含まれています。

DateTime クラスに基づく特定のタイムスタンププロパティは、ローカル時間に解析されていました。これらには、タイムスタンプの応答アンマーシャラーと、形式 TimestampFormat.ISO8601 および のタイムスタンプのリストが含まれていましたTimestampFormat.RFC822。代わりに UTC 時間を返すように DateTime 解析が更新されました。

ConvertFromUnixEpochSeconds および ConvertFromUnixEpochMilliseconds

Unix エポック秒を DateTime 構造に変換する ConvertFromUnixEpochSeconds メソッドと ConvertFromUnixEpochMilliseconds メソッドは、Unix エポック時間を UTC 時間ではなくローカル時間として返していました。 DateTime これらのメソッドは UTC 時間を返すようになりました。

ログ記録

SDK でログ記録を有効にする方法が V4 用に更新されました。コンソールおよびシステム診断へのログ記録は V3 と同じように機能します。つまり、AWSConfigs クラスの LoggingConfig.LogToプロパティを LoggingOptions.Consoleまたは に設定しますLoggingOptions.SystemDiagnostics。の LoggingOptions オプションlog4netは、リフレクションを使用して のインメモリインスタンスにアタッチするための SDK の内部ロジックとともに削除されましたlog4net。

SDK のログ記録フレームワークへのログインを含めるために、別のアダプターパッケージを使用して SDK をログ記録フレームワークに接続します。には AWSSDK.Extensions.Logging.Log4NetAdaptor パッケージlog4netを使用し、 には AWSSDK.Extensions.Logging.ILoggerAdaptor パッケージを使用しますMicrosoft.Extensions.Logging。次のコード例は、これら 2 つのケースでログ記録を設定する方法を示しています。

log4net の設定例

AWSSDK.Extensions.Logging.Log4NetAdaptor NuGet パッケージを追加し、 から静的ConfigureAWSSDKLoggingメソッドを呼び出します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));
```

Microsoft.Extensions.Logging の設定例

AWSSDK.Extensions.Logging.ILoggerAdaptor NuGet パッケージを追加し、ILoggerFactoryインターフェイスからConfigureAWSSDKLogging拡張機能メソッドを呼び出します。

```
var builder = WebApplication.CreateBuilder(args);

var app = builder.Build();

app.Services.GetRequiredService<ILoggerFactory>()
    .ConfigureAWSSDKLogging();
```

HTTP 2 のサポート

双方向ストリーミングを有効にするために HTTP 2 のサポートが追加されました。詳細については、「HTTP 2 のサポート」を参照してください。

シングルサインオン

SSOAWSCredentialsOptions クラスの SupportsGettingNewTokenプロパティのデフォルト値がから trueに変更されましたfalse。SSO 認証情報を取得するために SSOAWSCredentials クラスを使用するアプリケーションがある場合は、 Options.SupportsGettingNewTokenプロパティを に設定する必要がありますtrue。この設定の例については、「」のコード例を参照してください.NET アプリケーションのみを使用する SSO のチュートリアル。詳細については、aws-sdk-net GitHub リポジトリの「PR 3737」を参照してください。

DynamoDB に固有の変更

以下の変更は Amazon DynamoDB に固有のものです。その多くは重大な変更です。

の V4 での DynamoDB の変更の詳細については AWS SDK for .NET、ブログ記事V4 の AWS SDK for .NET プレビュー 4」を参照してください。

を開いて項目を表示する

SDK for DynamoDB の V4 の変更は、テスト可能性に関するいくつかの問題に対処しますが、主に高レベルライブラリを中心にしています。

- コード内の DocumentModel という名前DocumentModel.NET ドキュメントモデル。
- コード内の DataModel という名前の .NET オブジェクト永続性モデル。

これらのプログラミングモードの詳細については、このガイドDynamoDBの「」を参照してください。

ドキュメントモデル: モック**IAmazonDynamoDB**インターフェイスの例外を更新

SDK の V4 より前のドキュメントモデルでは、テーブルがモックされた IAmazonDynamoDB インターフェイスで初期化された場合、 が返されますNullReferenceException。InvalidOperationException 代わりに SDK の V4 が返されます。非同期Tableメソッドはモッククライアントで動作しますが、 から同期メソッドを呼び出すと例外が表示される場合があります.NET/Core/Standard。

この変更の詳細については、GitHub の「PR 3388」を参照してください。

ドキュメントモデル: **FromJson**メソッドと**ToJson**メソッド

ドキュメントクラスの FromJsonおよび ToJsonメソッドは、シリアル化に LitJson System.Text.Jsonの代わりに を使用するようになり、LitJson は SDK の V4 から削除されました。を使用する利点System.Text.Jsonは、このパーサーが数値浮動小数点プロパティのより高い精度をサポートする .NET Decimalタイプの使用をサポートしていることです。

オブジェクト永続性モデル: **DynamoDBOperationConfig** クラス

オブジェクト永続性モデルでは、共有 DynamoDBOperationConfig クラスに次の変更が行われました。

- クラスは、SaveConfig、LoadConfig、QueryConfig などの新しいオペレーション固有のクラスに分割されています。にかかる方法は廃止とマークDynamoDBOperationConfigされ、今後削除される可能性があります。

  この変更の詳細については、GitHub の「PR 3421」を参照してください。
- MetadataCachingMode および DisableFetchingTableMetadataプロパティが クラスから削除されました。これらのプロパティは、前述の新しいオペレーション固有のクラスに含まれて

いませんでした。削除されたプロパティはテーブルレベルの設定であり、AWSConfigsDynamoDB クラスのグローバルContextプロパティまたは DynamoDBContextConfig クラスで指定する必要があります。

この変更の詳細については、GitHub の「PR 3422」を参照してください。

- クラスは DynamoDBContextConfig クラスから継承されなくなりました。これにより、DynamoDBContext のコンストラクタにDynamoDBOperationConfigオブジェクトを渡すことができなくなり、オペレーション固有の設定の一部のプロパティ (などOverrideTableName) は適用されません。

  この変更の詳細については、GitHub の「PR 3422」を参照してください。

オブジェクト永続性モデル: 多型

DynamoDBPolymorphicTypeAttribute クラスがオブジェクト永続性モデルに追加されました。このクラスにより、多型タイプのシリアル化と逆シリアル化がサポートされます。詳細については、GitHub の「PR 3643」を参照してください。

ドキュメントモデルとオブジェクト永続性モデル: モック可能なオペレーション

新しいオペレーション固有のインターフェイスが追加され、お客様が DynamoDB オペレーションをモックできるようになりました。IDynamoDBContext インターフェイスのファクトリメソッドが更新され、新しいインターフェイスが返されました。

この変更の詳細については、GitHub の「PR 3450」を参照してください。

- オブジェクト永続性モデル
  - IBatchGet および IMultiTableBatchGetインターフェイスを介したモックBatchGetオペレーション。
  - IBatchWrite および IMultiTableBatchWriteインターフェイスを介したモックBatchWriteオペレーション。
  - ITransactGet および IMultiTableTransactGetインターフェイスを介したモックTransactGetオペレーション。
  - ITransactWrite および IMultiTableTransactWriteインターフェイスを介したモックTransactWriteオペレーション。
  - IAsyncSearch インターフェイスを介して Scanおよび Queryオペレーションをモックします。
- ドキュメントモデル

- ITable インターフェイスを介したモックTableオペレーション。
- ISearch インターフェイスを介して Scanと Queryオペレーションをモックします。
- IDocumentTransactWrite および IMultiTableDocumentTransactWriteインターフェイスを介したモックTransactWriteオペレーション。
- IDocumentTransactGet および IMultiTableDocumentTransactGetインターフェイスを介したモックTransactGetオペレーション。
- IDocumentBatchWrite および IMultiTableDocumentBatchWriteインターフェイスを介したモックBatchWriteオペレーション。
- IDocumentBatchGet および IMultiTableDocumentBatchGetインターフェイスを介したモックBatchGetオペレーション。

ドキュメントモデルとオブジェクト永続性モデル: ネイティブ AOT のサポート

ネイティブ AOT の制限は、ネストされた .NET タイプのサポートです。場合によっては、これらのネストされた型が .NET コンパイラのトリミングコンポーネントによって認識されないことがあります。この場合、System.InvalidOperationException: Type <type> is unsupported, it cannot be instantiated.「」などの例外が表示されることがあります。

この制限を回避するには、サブタイプへの依存関係をトリマーに通知する をコードパスのDynamicDependencyどこかに追加します。最上位の .NET タイプのコンストラクタが保存されている可能性があります。次のコード例は、 DynamicDependency 属性の使用方法を示しています。

```
[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; }
}
```

DynamoDBStreams

DynamoDBStreams は AWSSDK.DynamoDB NuGet パッケージから削除され、独自のパッケージ AWSSDK.DynamoDBStreams で利用でき、独自の名前空間 がありますAmazon.DynamoDBStreams。

`TableNamePrefix` 値の削除を許可する

個々のオペレーションレベルで DynamoDBContextConfig クラスで TableNamePrefixプロパティの値を削除できるようになりました。この変更の詳細については、GitHub の「PR 3476」を参照してください。

`RetrieveDateTimeInUtc` プロパティ

DynamoDBContextConfig クラスでは、 RetrieveDateTimeInUtcプロパティのデフォルト値が に変更されましたtrue。

`DynamoDBContextTableNamePrefix` プロパティ

AWSConfigsDynamoDB クラスから DynamoDBContextTableNamePrefixプロパティを削除しました。ユーザーは AWSConfigsDynamoDB.Context.TableNamePrefixの代わりに を呼び出す必要があります

EC2 に固有の変更

以下の変更は Amazon EC2 に固有のものです。そのほとんどまたはすべてが重大な変更です。

を開いて項目を表示する

`GetDecryptedPassword`

Amazon EC2 のGetDecryptedPassword拡張機能は、AWSSDK.Extensions.EC2.DecryptPassword という別の拡張機能パッケージに移動されました。この変更は、OpenSSL 3 をサポートし、BouncyCastle.Cryptography に依存するためです。

このパッケージのソースコードは GitHub の https://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.EC2.DecryptPassword にあります。

Amazon EC2 IMDSv1 のサポート

インスタンスメタデータサービスバージョン 1 (IMDSv1) のサポートが削除されました。SDK の V4 は、IMDS から認証情報やその他のメタデータを取得するときに、常にインスタンスメタデータサービスバージョン 2 (IMDSv2) を使用します。IMDS の詳細については、Amazon EC2 ユーザーガイド」の「IMDS の使用」を参照してください。

変更または削除された要素のプログラミング

- `Amazon.EC2.Import` 名前空間とコード全体が削除されました。
- `Amazon.EC2.Util` 名前空間とコード全体が削除されました。これには、Windows 用の EC2 AMI の検索に使用された AMIs ユーティリティが含まれます。
- 古い`IpRanges`プロパティが IpPermission クラスから削除されました。代わりに `Ipv4Ranges`または `Ipv6Ranges`プロパティを使用します。
- ECEC2InstanceMetadata クラスから次の古いフィールドが削除されました: `EC2_METADATA_SVC`、`EC2_METADATA_ROOT`、`EC2_USERDATA_ROOT`、`EC2_DYNAMICDATA_ROOT`、`EC`

S3 に固有の変更

以下の変更は Amazon S3 に固有のものです。そのほとんどまたはすべてが重大な変更です。

を開いて項目を表示する

AWS リージョン us-east-1

`us-east-1` リージョン用に設定された Amazon S3 サービスクライアントは、他のリージョンのバケットにアクセスできなくなります。バケットには、バケットがあるリージョン用に設定された S3 サービスクライアントを使用してアクセスする必要があります。

この変更の詳細については、ブログ記事「V4 の AWS SDK for .NET プレビュー 4」を参照してください。

S3 暗号化クライアント

`Amazon.S3.Encryption` 名前空間で定義されている Amazon S3 暗号化クライアントが AWSSDK.S3 パッケージから削除されました。このクライアントは Amazon.Extensions.S3.Encryption という独自のパッケージに移動され、そのドキュメントは にありますhttps://aws.github.io/amazon-s3-encryption-client-dotnet/api/

Amazon.Extensions.S3.Encryption.html。移行に関する詳細については、「the section called "S3 暗号化クライアントの移行 (V1 から V2)"」を参照してください。S3 暗号化の詳細については、「Amazon S3 暗号化クライアントデベロッパーガイド」の「サポートされている暗号化アルゴリズムAmazon S3」を参照してください。

の S3 タグ付けディレクティブ **CopyObject**

TaggingDirective プロパティは、 AmazonS3Client.CopyObjectメソッドで使用される CopyObjectRequest クラスのパブリックプロパティとして公開されています。このプロパティは、CopyObject アクションで定義されている Amazon S3 x-amz-tagging-directiveパラメータに対応します。

タグ付けディレクティブは COPY に自動的に設定されなくなりました。開発者がタグ付けディレクティブを指定しない場合、S3 バックエンドは自動的にそれを COPY と想定しますが、開発者が明示的にプロパティを null に設定した場合、値はまったく設定されません。

S3 設定の **UseArnRegion**プロパティ

Amazon.S3.AmazonS3Config クラスの UseArnRegionプロパティが更新され、AWS_S3_USE_ARN_REGION環境変数が共有 AWS configファイルの s3_use_arn_region設定よりも優先されるようになりました。これらの変数と設定の詳細については、 SDK およびツールリファレンスガイドの「設定リファレンス」を参照してください。 AWS SDKs

メソッド**CopyObject**と **CopyPart**メソッドの先頭のスラッシュ

Amazon S3 CopyObjectおよび CopyPartメソッドでは、先頭のスラッシュはトリミングされなくなります。DisableTrimmingLeadingSlash プロパティが CopyObjectRequest クラスと CopyPartRequest クラスから削除されました。

DoesS3BucketExist... メソッド

古い メソッドDoesS3BucketExistと DoesS3BucketExistAsyncメソッドは、ICoreAmazonS33 インターフェイスを実装する AmazonS3Util クラスから削除されました。これらのメソッドは、常に HTTP を使用するため削除されました。代わりに DoesS3BucketExistV2 と DoesS3BucketExistV2Async を使用します。

SDK は常に SigV4 を使用します

のバージョン 4 では、署名リクエストに AWS SDK for .NET 常に AWS 署名バージョン 4 (SigV4) が使用されます。この変更により、次の関連変更が発生します。

- AWSConfigsS3 クラスの UseSignatureVersion4プロパティは削除されました。
- Amazon.Runtime.ClientConfig クラスの SignatureVersionプロパティは削除されました。このプロパティは、下位互換性のために Amazon S3 でのみ使用されました。
- RegionEndpoint.Endpoint クラスは削除されました。これには、Amazon S3 の署名バージョンを上書きするために使用された SignatureVersionOverrideプロパティが含まれます。代わりに、サービス固有のclient.DetermineServiceOperationEndPoint()方法を使用します。
- SigV4 を使用するように AmazonS3Util.PostUpload および S3PostUploadSignedPolicy.GetSignedPolicy メソッドを更新しました。その結果、S3PostUploadSignedPolicy.GetSignedPolicyV4メソッドは、 が同じ関数を実行するGetSignedPolicyようになったため削除されました。さらに、 GetSignedPolicyにはリージョンエンドポイントの 3 番目のパラメータが与えられています。

GetACL および PutACLメソッド

AmazonS3Client クラスの GetACLおよび PutACLメソッドは廃止とマークされています。これらのメソッドの機能にアクセスするには、代わりに GetBucketACL、、PutBucketACL、GetObjectACLおよび の新しいメソッドを使用しますPutObjectACL。

廃止されたプログラミング要素の削除

列挙値、型、メソッド、名前空間など、Amazon S3 実装の多くのプログラミング要素が SDK の V4 から削除されました。これらは、以前に説明されていない場合は、削除に対応するために実行できる潜在的な手順とともに、以下に一覧表示されます。

- DisableMD5Stream プロパティが TransferUtilityUploadRequest クラスから削除されました。代わりに、DisableDefaultChecksumValidation プロパティを使用してください。

  さらに、 CalculateContentMD5Headerプロパティは TransferUtilityUploadRequest クラスから削除されました。SDK はデフォルトでチェックサムを計算するため、このプロパティは不要になりました。
- ServerSideEncryptionMethod および ServerSideEncryptionKeyManagementServiceKeyIdプロパティは CopyPartRequest クラスから削除されました。代わりに、InitiateMultipartUploadRequest クラスで同じ名前のプロパティを使用します。これは、AmazonS3Client クラスの一部のInitiateMultipartUpload...メソッドで使用されます。

- Expires プロパティが GetObjectResponse クラスから削除されました。代わりに、ExpiresString プロパティを使用してください。文字列が有効なタイムスタンプ形式ではない可能性があるため、コードは に変換するときに TryParseメソッドを使用する必要がありますDateTime。
- 古い AWS リージョン 識別子が S3Region 列挙から削除されました。
- Prefix プロパティが LifecycleRule クラスから削除されました。代わりに、Filter プロパティを使用してください。

  さらに、 NoncurrentVersionTransitionおよび Transitionプロパティは LifecycleRule クラスから削除されました。代わりに NoncurrentVersionTransitionsおよび Transitionsコレクションを使用します。
- Event プロパティが TopicConfiguration クラスから削除されました。代わりに Eventsコレクションを使用します。
- CalculateContentMD5Header プロパティ。SDK はデフォルトでチェックサムを計算するため、このプロパティを設定する必要がなくなりました。
- Bucket プロパティが SelectObjectContentRequest クラスから削除されました。代わりに、BucketName プロパティを使用してください。
- NumberOfUploadThreads プロパティが TransferUtilityConfig クラスから削除されました。代わりに ConcurrentServiceRequestsプロパティを使用します。

削除された要素のプログラミング

列挙値、型、メソッド、名前空間など、多くのプログラミング要素が SDK の V4 から削除されました。これらは、以前に説明されていない場合は、削除に対応するために実行できる潜在的な手順とともに、以下に一覧表示されます。

を開いて項目を表示する

Amazon.Auth.AccessControlPolicy.ActionIdentifiers 名前空間

Amazon.Auth.AccessControlPolicy.ActionIdentifiers 名前空間は削除されました。これには、 IdentityandAccessManagementActionIdentifiers クラスで定義された IAM アクション識別子が含まれます。これらのアクション識別子を使用するコードは、アクション名の文字列値を使用するように変更する必要があります。

詳細については、「IAM ユーザーガイド」のthe section called “JSON からのマネージドポリシーの作成”「」および「JSON ポリシーの概要」を参照してください。 https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html#access_policies-json

`ClientConfig` クラス

Amazon.Runtime.ClientConfig クラスは、AmazonS3Config などのサービスクライアント設定クラスのベースクラスです。このクラスから次のプログラミング要素が削除されました。

- `DetermineServiceURL` および `DetermineDnsSuffix`メソッドは削除されました。代わりに、AmazonS3Client.DetermineServiceOperationEndpoint などのサービスクライアントの `DetermineServiceOperationEndpoint`メソッドを使用します。
- `ReadEntireResponse` プロパティは削除されました。代わりに次のいずれかを使用します。
  - AWSConfigs.LoggingConfig クラスの `LogResponses`プロパティ。
  - クライアント設定の `LogResponse`プロパティ。AmazonS3Config など。

`Amazon.Runtime` 名前空間

Amazon.Runtime 名前空間が次のように更新されました。

- 古い`ECSTaskCredentials`クラスが名前空間から削除されました。代わりに、Amazon EKS Pod ID もサポートする GenericContainerCredentials プロバイダーを使用してください。
- 古い `StoredProfileAWSCredentials`および `StoredProfileCredentials` クラスが名前空間から削除されました。代わりに、Amazon.Runtime.CredentialManagement 名前空間の NetSDKCredentialsFile または SharedCredentialsFile クラスを使用します。
- SSOAWSCredentials クラスの古い`HasCachedAccessTokenAvailable`メソッドが名前空間から削除されました。
- 古い`EnvironmentAWSCredentials`クラスが名前空間から削除されました。代わりに AppConfigAWSCredentials クラスを使用します。
- 古い`StoredProfileFederatedCredentials`クラスが名前空間から削除されました。代わりに FederatedAWSCredentials クラスを使用します。
- 名前空間から次の古いクラスが削除されました: `EnvironmentVariableAWSEndpointDiscoveryEnabled`、`ProfileAWSEndpointDiscoveryEnab`
- 古い`UseSigV4`プロパティが AmazonWebServiceRequest クラスから削除されました。代わりに、`SignatureVersion` プロパティを使用してください。

- Amazon.Runtime.Internal.Util 名前空間の ProfileIniFile クラスには、 というオーバーロードされたメソッドがありますTryGetSection。の outパラメータをサポートしていないメソッドのバージョンnestedPropertiesが クラスから削除されました。
- Amazon.Runtime.Internal.Auth 名前空間の古いEventBridgeSignerクラスは削除されました。
- 古いParametersディクショナリプロパティが WebServiceRequestEventArgs クラスから削除されました。代わりに、ParameteCollection プロパティを使用してください。

BouncyCastle

BouncyCastle のソースコピーが SDK の V4 から削除されました。

StoredProfileSAMLCredentials クラス

Amazon.SecurityToken.SAML 名前空間の古いStoredProfileSAMLCredentialsクラスは削除されました。代わりに、Amazon.Runtime 名前空間で FederatedAWSCredentials クラスを使用します。

AWSSDKUtils クラス

AWSSDKUtils クラスから、ResolveResourcePath、ProtectEncodedSlashUrlEncode、および メソッドが削除されましたConvertToUnixEpochMilliSeconds。

ProfileManager クラス

古いProfileManagerクラスが Amazon.Util 名前空間から削除されました。代わりに、Amazon.Runtime.CredentialManagement 名前空間の NetSDKCredentialsFile または SharedCredentialsFile クラスを使用します。

AWSConfigs クラス

次の古いプロパティが AWSConfigs クラスから削除されました:
Logging、ResponseLogging、LogMetrics。代わりに、LoggingConfig プロパティを使用してください。

ConditionFactory クラス

次の署名を持つメソッドが ConditionFactory クラス から削除されましたNewCondition(ConditionFactory.DateComparisonType, DateTime)。代わりに NewConditionUtc メソッドを使用します。

Amazon CloudFront ユーティリティ

古い`Amazon.CloudFront.Util`名前空間と`AmazonCloudFrontUtil`クラスは削除されました。

AWS IoT

ListPrincipalThingsResponse クラスでは、ページ分割を優先して、`NextToken`オーバーライドのレガシーカスタマイズが削除されました。

AWS Lambda

AmazonLambdaClient クラスの次の`Invoke...`メソッドは、名前がわかりにくいため削除されました。

- 次の署名を持つ V3 メソッドが削除されました: `InvokeAsyncResponse InvokeAsync(InvokeAsyncRequest)`。これは SDK の V3 の同期メソッドです。代わりに `InvokeResponse Invoke(InvokeRequest)` (同期処理の場合) または `Task InvokeAsync(InvokeRequest, CancellationToken)` (非同期処理の場合) を使用します。
- 次の署名を持つ V3 メソッドが削除されました: `Task InvokeAsyncAsync(InvokeAsyncRequest, CancellationToken)`。これは SDK の V3 の非同期メソッドです。代わりに `Task InvokeAsync(InvokeRequest, CancellationToken)` を使用します。

Amazon SageMaker ランタイム

PayloadPart クラスの古いコンストラクタが削除されました。

.NET Standard 1.3 からの移行

2019 年 6 月 27 日に、Microsoft は .NET Core 1.0 および .NET Core 1.1 バージョンのサポートを終了しました。この発表の後、 AWS は 2020 年 12 月 31 AWS SDK for .NET 日に で .NET Standard 1.3 のサポートを終了しました。

AWS は、2020 年 10 月 1 日まで、 AWS SDK for .NET ターゲット .NET Standard 1.3 でサービスの更新とセキュリティ修正を提供し続けました。その日以降、.NET Standard 1.3 ターゲットはメンテナンスモードに移行しました。つまり、新しい更新はリリースされませんでした。重要なバグ修正とセキュリティパッチのみ AWS を適用しました。

2020 年 12 月 31 日、 での .NET Standard 1.3 のサポートは終了 AWS SDK for .NET しました。この日以降、バグ修正やセキュリティパッチは適用されなくなりました。同バージョンを対象として構築されたアーティファクトは、NuGet で引き続きダウンロードできます。

必要な作業

- .NET Framework を使用してアプリケーションを実行している場合、これによる影響はありません。

- .NET Core 2.0 以降を使用してアプリケーションを実行している場合、これによる影響はありません。

- .NET Core 1.0 または .NET Core 1.1 を使用してアプリケーションを実行している場合は、Microsoft の移行手順に従ってアプリケーションを新しいバージョンの .NET Core に移行してください。.NET Core 3.1 以降を推奨します。

- 現時点ではアップグレードできないビジネスクリティカルなアプリケーションを実行している場合は、 AWS SDK for .NETの現在のバージョンを引き続き使用できます。

ご質問やご不明な点がある場合は、AWS サポートにお問い合わせください。

この AWS 製品またはサービスのセキュリティ

クラウドセキュリティは Amazon Web Services (AWS) の最優先事項です。 AWS のお客様は、セキュリティを非常に重視する組織の要件を満たせるように構築されたデータセンターとネットワークアーキテクチャーから利点を得ます。セキュリティは、 AWS とお客様の間の責任共有です。責任共有モデルでは、これをクラウドのセキュリティおよびクラウド内のセキュリティとして説明しています。

クラウドのセキュリティ – AWS クラウドで提供されるすべてのサービスを実行するインフラストラクチャ AWS を保護し、安全に使用できるサービスを提供します。における当社のセキュリティ責任は最優先事項であり AWS、当社のセキュリティの有効性は、AWS コンプライアンスプログラムの一環としてサードパーティーの監査者によって定期的にテストおよび検証されます。

クラウド内のセキュリティ – お客様の責任は、使用している AWS サービス、データの機密性、組織の要件、適用される法律や規制などのその他の要因によって決まります。

この AWS 製品またはサービスは、サポートする特定の Amazon Web Services (AWS) サービスを通じて責任共有モデルに従います。 AWS サービスセキュリティ情報については、AWS 「サービスセキュリティドキュメント」ページとAWS 、コンプライアンスプログラムによる AWS コンプライアンスの取り組みの対象となるサービスを参照してください。

トピック

- この AWS 製品またはサービスのデータ保護
- Identity and Access Management
- この AWS 製品またはサービスのコンプライアンス検証
- この AWS 製品またはサービスの耐障害性
- この AWS 製品またはサービスのインフラストラクチャセキュリティ
- で最小 TLS バージョンを適用する AWS SDK for .NET
- S3 暗号化クライアントの移行 (V1 から V2)
- S3 暗号化クライアントの移行 (V2 から V4)

この AWS 製品またはサービスのデータ保護

責任 AWS 共有モデル、この AWS 製品またはサービスのデータ保護に適用されます。このモデルで説明されているように、 AWS はすべての を実行するグローバルインフラストラクチャを保護する

責任があります AWS クラウド。ユーザーは、このインフラストラクチャでホストされるコンテンツに対する管理を維持する責任があります。また、使用する「 AWS のサービス 」のセキュリティ設定と管理タスクもユーザーの責任となります。データプライバシーの詳細については、データプライバシーに関するよくある質問を参照してください。欧州でのデータ保護の詳細については、AWS セキュリティブログに投稿された「AWS 責任共有モデルおよび GDPR」のブログ記事を参照してください。

データ保護の目的で、認証情報を保護し AWS アカウント 、 AWS IAM アイデンティティセンターまたは AWS Identity and Access Management (IAM) を使用して個々のユーザーを設定することをお勧めします。この方法により、それぞれのジョブを遂行するために必要な権限のみが各ユーザーに付与されます。また、次の方法でデータを保護することもお勧めします:

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。TLS 1.2 は必須ですが、TLS 1.3 を推奨します。
- で API とユーザーアクティビティのログ記録を設定します AWS CloudTrail。CloudTrail 証跡を使用して AWS アクティビティをキャプチャする方法については、「 AWS CloudTrail ユーザーガイド」のCloudTrail 証跡の使用」を参照してください。
- AWS 暗号化ソリューションと、その中のすべてのデフォルトのセキュリティコントロールを使用します AWS のサービス。
- Amazon Macie などの高度な管理されたセキュリティサービスを使用します。これらは、Amazon S3 に保存されている機密データの検出と保護を支援します。
- コマンドラインインターフェイスまたは API AWS を介して にアクセスするときに FIPS 140-3 検証済み暗号化モジュールが必要な場合は、FIPS エンドポイントを使用します。利用可能な FIPS エンドポイントの詳細については、「連邦情報処理規格 (FIPS) 140-3」を参照してください。

お客様の E メールアドレスなどの極秘または機密情報を、タグ、または [名前] フィールドなどの自由形式のテキストフィールドに含めないことを強くお勧めします。これは、コンソール、API、AWS CLIまたは SDK を使用して、この AWS 製品またはサービスまたは他の AWS のサービス を使用する場合も同様です。 AWS SDKs タグ、または名前に使用される自由記述のテキストフィールドに入力したデータは、請求または診断ログに使用される場合があります。外部サーバーに URL を提供する場合、そのサーバーへのリクエストを検証できるように、認証情報を URL に含めないことを強くお勧めします。

Identity and Access Management

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御 AWS のサービス するのに役立つ です。IAM 管理者は、誰を認証 (サインイン) し、誰に AWS リソースの使用を許可する (アクセス許可を付与する) かを制御します。IAM は、追加料金なしで使用できる AWS のサービス です。

トピック

- オーディエンス
- アイデンティティを使用した認証
- ポリシーを使用したアクセスの管理
- IAM AWS のサービス の操作方法
- AWS ID とアクセスのトラブルシューティング

オーディエンス

AWS Identity and Access Management (IAM) の使用方法は、 で行う作業によって異なります AWS。

サービスユーザー – AWS のサービス を使用してジョブを実行する場合、管理者から必要な認証情報とアクセス許可が提供されます。さらに多くの AWS 機能を使用して作業を行う場合は、追加のアクセス許可が必要になる場合があります。アクセスの管理方法を理解すると、管理者に適切なアクセス許可をリクエストするのに役に立ちます。の機能にアクセスできない場合は AWS、 AWS のサービス AWS ID とアクセスのトラブルシューティング「」または使用している のユーザーガイドを参照してください。

サービス管理者 – 社内の AWS リソースを担当している場合は、通常、 へのフルアクセスがあります AWS。サービスユーザーがどの AWS 機能やリソースにアクセスするかを決めるのは管理者の仕事です。その後、IAM 管理者にリクエストを送信して、サービスユーザーの権限を変更する必要があります。このページの情報を点検して、IAM の基本概念を理解してください。会社で IAM を使用する方法の詳細については AWS、使用している AWS のサービス のユーザーガイドを参照してください。

IAM 管理者 - 管理者は、 AWSへのアクセスを管理するポリシーの書き込み方法の詳細について確認する場合があります。IAM で使用できる AWS アイデンティティベースのポリシーの例を表示するには、 AWS のサービス 使用している のユーザーガイドを参照してください。

アイデンティティを使用した認証

認証とは、ID 認証情報 AWS を使用して にサインインする方法です。、IAM ユーザー AWS アカウントのルートユーザー、または IAM ロールを引き受けることで認証される必要があります。

AWS IAM アイデンティティセンター (IAM Identity Center)、シングルサインオン認証、Google/Facebook 認証情報などの ID ソースからの認証情報を使用して、フェデレーティッド ID としてサインインできます。サインインの詳細については、「AWS サインイン ユーザーガイド」の「AWS アカウントにサインインする方法」を参照してください。

プログラムによるアクセスの場合、 は SDK と CLI AWS を提供してリクエストを暗号化して署名します。詳細については、「IAM ユーザーガイド」の「API リクエストに対するAWS 署名バージョン 4」を参照してください。

AWS アカウント ルートユーザー

を作成するときは AWS アカウント、すべての AWS のサービス および リソースへの完全なアクセス権を持つ AWS アカウント ルートユーザーと呼ばれる 1 つのサインインアイデンティティから始めます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザー認証情報を必要とするタスクについては、「IAM ユーザーガイド」の「ルートユーザー認証情報が必要なタスク」を参照してください。

フェデレーテッドアイデンティティ

ベストプラクティスとして、人間のユーザーが一時的な認証情報 AWS のサービス を使用して にアクセスするには、ID プロバイダーとのフェデレーションを使用する必要があります。

フェデレーティッド ID は、エンタープライズディレクトリ、ウェブ ID プロバイダー、または ID Directory Service ソースの認証情報 AWS のサービス を使用して にアクセスするユーザーです。フェデレーテッドアイデンティティは、一時的な認証情報を提供するロールを引き受けます。

アクセスを一元管理する場合は、 AWS IAM アイデンティティセンターをお勧めします。詳細については、「AWS IAM アイデンティティセンター ユーザーガイド」の「IAM アイデンティティセンターとは」を参照してください。

IAM ユーザーとグループ

IAM ユーザーは、特定の個人やアプリケーションに対する特定のアクセス許可を持つアイデンティティです。長期認証情報を持つ IAM ユーザーの代わりに一時的な認証情報を使用することをお勧め

します。詳細については、IAM ユーザーガイドの「ID プロバイダーとのフェデレーションを使用して にアクセスする必要がある AWS」を参照してください。

IAM グループは、IAM ユーザーの集合を指定し、大量のユーザーに対するアクセス許可の管理を容易にします。詳細については、「IAM ユーザーガイド」の「IAM ユーザーに関するユースケース」を参照してください。

IAM ロール

IAM ロールは、特定のアクセス許可を持つアイデンティであり、一時的な認証情報を提供します。ユーザーから IAM ロール (コンソール) に切り替えるか、 または API オペレーションを呼び出すことで、ロールを引き受けることができます。 AWS CLI AWS 詳細については、「IAM ユーザーガイド」の「ロールを引き受けるための各種方法」を参照してください。

IAM ロールは、フェデレーションユーザーアクセス、一時的な IAM ユーザーのアクセス許可、クロスアカウントアクセス、クロスサービスアクセス、および Amazon EC2 で実行するアプリケーションに役立ちます。詳細については、IAM ユーザーガイド の IAM でのクロスアカウントリソースアクセス を参照してください。

ポリシーを使用したアクセスの管理

でアクセスを制御する AWS には、ポリシーを作成し、ID AWS またはリソースにアタッチします。ポリシーは、アイデンティティまたはリソースに関連付けられている場合のアクセス許可を定義します。 は、プリンシパルがリクエストを行うときにこれらのポリシー AWS を評価します。ほとんどのポリシーは JSON ドキュメント AWS として に保存されます。JSON ポリシードキュメントの詳細については、「IAM ユーザーガイド」の「JSON ポリシー概要」を参照してください。

管理者は、ポリシーを使用して、どのプリンシパルがどのリソースに対して、どのような条件でアクションを実行できるかを定義することで、誰が何にアクセスできるかを指定します。

デフォルトでは、ユーザーやロールにアクセス許可はありません。IAM 管理者は IAM ポリシーを作成してロールに追加し、このロールをユーザーが引き受けられるようにします。IAM ポリシーは、オペレーションの実行方法を問わず、アクセス許可を定義します。

アイデンティティベースのポリシー

アイデンティティベースのポリシーは、アイデンティティ (ユーザー、グループ、またはロール) にアタッチできる JSON アクセス許可ポリシードキュメントです。これらのポリシーは、アイデンティティがどのリソースに対してどのような条件下でどのようなアクションを実行できるかを制御し

ます。アイデンティティベースポリシーの作成方法については、IAM ユーザーガイド の カスタマー管理ポリシーでカスタム IAM アクセス許可を定義する を参照してください。

アイデンティティベースのポリシーは、インラインポリシー (単一の ID に直接埋め込む) または管理ポリシー (複数の ID にアタッチされたスタンドアロンポリシー) にすることができます。管理ポリシーとインラインポリシーのいずれかを選択する方法については、「IAM ユーザーガイド」の「管理ポリシーとインラインポリシーのいずれかを選択する」を参照してください。

リソースベースのポリシー

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。例としては、IAM ロール信頼ポリシーや Amazon S3 バケットポリシーなどがあります。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーでは、IAM の AWS マネージドポリシーを使用できません。

アクセスコントロールリスト (ACL)

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするためのアクセス許可を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon S3、および Amazon VPC は AWS WAF、ACLs。ACL の詳細については、Amazon Simple Storage Service デベロッパーガイド の アクセスコントロールリスト (ACL) の概要 を参照してください。

その他のポリシータイプ

AWS は、より一般的なポリシータイプによって付与されるアクセス許可の上限を設定できる追加のポリシータイプをサポートしています。

- アクセス許可の境界 – アイデンティティベースのポリシーで IAM エンティティに付与することのできるアクセス許可の数の上限を設定します。詳細については、「IAM ユーザーガイド」の「IAM エンティティのアクセス許可境界」を参照してください。
- サービスコントロールポリシー (SCP) - AWS Organizations内の組織または組織単位の最大のアクセス許可を指定します。詳細については、「AWS Organizations ユーザーガイド」の「サービスコントロールポリシー」を参照してください。

- リソースコントロールポリシー (RCP) – は、アカウント内のリソースで利用できる最大数のアクセス許可を定義します。詳細については、「AWS Organizations ユーザーガイド」の「リソースコントロールポリシー (RCP)」を参照してください。
- セッションポリシー – ロールまたはフェデレーションユーザーの一時セッションを作成する際にパラメータとして渡される高度なポリシーです。詳細については、「IAM ユーザーガイド」の「セッションポリシー」を参照してください。

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成されるアクセス許可を理解するのがさらに難しくなります。が複数のポリシータイプが関与する場合にリクエストを許可するかどうか AWS を決定する方法については、「IAM ユーザーガイド」の「ポリシー評価ロジック」を参照してください。

IAM AWS のサービス の操作方法

ほとんどの IAM 機能と AWS のサービス の連携方法の概要については、「IAM ユーザーガイド」のAWS 「IAM と連携する のサービス」を参照してください。

IAM AWS のサービス で特定の を使用する方法については、関連するサービスのユーザーガイドのセキュリティセクションを参照してください。

AWS ID とアクセスのトラブルシューティング

次の情報は、 および IAM の使用時に発生する可能性がある一般的な問題の診断 AWS と修復に役立ちます。

トピック

- でアクションを実行する権限がありません AWS
- iam:PassRole を実行する権限がありません
- 自分の 以外のユーザーに自分の AWS リソース AWS アカウント へのアクセスを許可したい

でアクションを実行する権限がありません AWS

アクションを実行する権限がないというエラーが表示された場合は、そのアクションを実行できるようにポリシーを更新する必要があります。

次のエラー例は、mateojackson IAM ユーザーがコンソールを使用して、ある *my-example-widget* リソースに関する詳細情報を表示しようとしたことを想定して、その際に必要な awes:*GetWidget* アクセス許可を持っていない場合に発生するものです。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

この場合、awes:*GetWidget* アクションを使用して *my-example-widget* リソースへのアクセスを許可するように、mateojackson ユーザーのポリシーを更新する必要があります。

サポートが必要な場合は、 AWS 管理者にお問い合わせください。サインイン認証情報を提供した担当者が管理者です。

iam:PassRole を実行する権限がありません

iam:PassRole アクションを実行する権限がないというエラーが表示された場合は、ポリシーを更新して AWSにロールを渡すことができるようにする必要があります。

一部の AWS のサービス では、新しいサービスロールまたはサービスにリンクされたロールを作成する代わりに、そのサービスに既存のロールを渡すことができます。そのためには、サービスにロールを渡す権限が必要です。

以下の例のエラーは、marymajor という IAM ユーザーがコンソールを使用して AWSでアクションを実行しようとする場合に発生します。ただし、このアクションをサービスが実行するには、サービスロールから付与された権限が必要です。Mary には、ロールをサービスに渡すアクセス許可がありません。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

この場合、Mary のポリシーを更新してメアリーに iam:PassRole アクションの実行を許可する必要があります。

サポートが必要な場合は、 AWS 管理者にお問い合わせください。サインイン資格情報を提供した担当者が管理者です。

自分の 以外のユーザーに自分の AWS リソース AWS アカウント へのアクセスを許可したい

他のアカウントのユーザーや組織外の人が、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまた

はアクセスコントロールリスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください:

- がこれらの機能 AWS をサポートしているかどうかを確認するには、「」を参照してくださいIAM AWS のサービス の操作方法。
- 所有 AWS アカウント している のリソースへのアクセスを提供する方法については、「IAM ユーザーガイド」の「所有 AWS アカウント している別の の IAM ユーザーへのアクセスを提供する」を参照してください。
- リソースへのアクセスをサードパーティーに提供する方法については AWS アカウント、IAM ユーザーガイドの「サードパーティー AWS アカウント が所有する へのアクセスを提供する」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、IAM ユーザーガイド の 外部で認証されたユーザー (ID フェデレーション) へのアクセスの許可 を参照してください。
- クロスアカウントアクセスにおけるロールとリソースベースのポリシーの使用方法の違いについては、IAM ユーザーガイド の IAM でのクロスアカウントのリソースへのアクセス を参照してください。

この AWS 製品またはサービスのコンプライアンス検証

AWS のサービス が特定のコンプライアンスプログラムの範囲内にあるかどうかを確認するには、「コンプライアンスAWS のサービス プログラムによるスコープ」の「コンプライアンス」を参照して、関心のあるコンプライアンスプログラムを選択します。一般的な情報については、AWS 「コンプライアンスプログラム」を参照してください。

を使用して、サードパーティーの監査レポートをダウンロードできます AWS Artifact。詳細については、「Downloading Reports in AWS Artifact」を参照してください。

を使用する際のお客様のコンプライアンス責任 AWS のサービス は、お客様のデータの機密性、貴社のコンプライアンス目的、適用される法律および規制によって決まります。を使用する際のコンプライアンス責任の詳細については AWS のサービス、AWS 「 セキュリティドキュメント」を参照してください。

この AWS 製品またはサービスは、サポートする特定の Amazon Web Services (AWS) サービスを通じて責任共有モデルに従います。 AWS サービスセキュリティ情報については、AWS 「サービスセキュリティドキュメント」ページとAWS 、コンプライアンスプログラムによる AWS コンプライアンスの取り組みの対象となるサービスを参照してください。

この AWS 製品またはサービスの耐障害性

AWS グローバルインフラストラクチャは、 AWS リージョン およびアベイラビリティーゾーンを中心に構築されています。

AWS リージョン は、複数の物理的に分離および分離されたアベイラビリティーゾーンを提供し、低レイテンシー、高スループット、および高度に冗長なネットワークで接続されます。

アベイラビリティーゾーンでは、ゾーン間で中断することなく自動的にフェールオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性、フォールトトレランス、および拡張性が優れています。

AWS リージョンとアベイラビリティーゾーンの詳細については、AWS 「 グローバルインフラストラクチャ」を参照してください。

この AWS 製品またはサービスは、サポートする特定の Amazon Web Services (AWS) サービスを通じて責任共有モデルに従います。 AWS サービスセキュリティ情報については、AWS 「サービスセキュリティドキュメント」ページとAWS 、コンプライアンスプログラムによる AWS コンプライアンスの取り組みの対象となるサービスを参照してください。

この AWS 製品またはサービスのインフラストラクチャセキュリティ

この AWS 製品またはサービスはマネージドサービスを使用するため、 グローバルネットワークセキュリティによって AWS 保護されています。 AWS セキュリティサービスと がインフラストラクチャ AWS を保護する方法については、AWS 「 クラウドセキュリティ」を参照してください。インフラストラクチャセキュリティのベストプラクティスを使用して AWS 環境を設計するには、「Security Pillar AWS Well-Architected Framework」の「Infrastructure Protection」を参照してください。

AWS 公開された API コールを使用して、ネットワーク経由でこの AWS 製品またはサービスにアクセスします。クライアントは次をサポートする必要があります。

- Transport Layer Security (TLS)。TLS 1.2 が必須で、TLS 1.3 をお勧めします。
- DHE (楕円ディフィー・ヘルマン鍵共有) や ECDHE (楕円曲線ディフィー・ヘルマン鍵共有) などの完全前方秘匿性 (PFS) による暗号スイート。これらのモードは Java 7 以降など、ほとんどの最新システムでサポートされています。

また、リクエストにはアクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。または、AWS Security Token Service (AWS STS) を使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

この AWS 製品またはサービスは、サポートする特定の Amazon Web Services (AWS) サービスを通じて責任共有モデルに従います。 AWS サービスセキュリティ情報については、AWS 「サービスセキュリティドキュメント」ページとAWS 、コンプライアンスプログラムによる AWS コンプライアンスの取り組みの対象となるサービスを参照してください。

で最小 TLS バージョンを適用する AWS SDK for .NET

AWS サービスと通信する際のセキュリティを強化するには、TLS 1.2 以降を使用する AWS SDK for .NET ように を設定する必要があります。

AWS SDK for .NET は、基盤となる .NET ランタイムを使用して、使用するセキュリティプロトコルを決定します。現行バージョンの .NET は、デフォルトでオペレーティングシステムがサポートする最新の設定済みプロトコルを使用します。この SDK の動作はアプリケーションで上書きできますが、上書きは推奨されません。

.NET Core

.NET Core は、デフォルトでオペレーティングシステムがサポートする最新の設定済みプロトコルを使用します。 AWS SDK for .NET は、この動作を上書きする機構を提供していません。

バージョン 2.1 より前の .NET Core を使用している場合は、.NET Core バージョンをアップグレードすることを強くお勧めします。

オペレーティングシステムごとの固有の情報については、以下を参照してください。

Windows

Windows の最新のディストリビューションでは、TLS 1.2 のサポートが デフォルトで有効になっています。Windows 7 SP1 または Windows Server 2008 R2 SP1 上で実行している場合は、レジストリで TLS 1.2 のサポートが有効になっていることを確認する必要があります。詳細については、https://learn.microsoft.com/ja-jp/windows-server/security/tls/tls-registry-settings#tls-12 を参照してください。以前のディストリビューションを実行している場合は、オペレーティングシステムをアップグレードする必要があります。WindowsのTLS 1.3サポートについては、必要最小限のクライアントまたはサーバーのバージョンについて、最新のMicrosoftドキュメントを確認してください。

macOS

.NET Core 2.1 以降を実行している場合、TLS 1.2 はデフォルトで有効になっています。TLS 1.2 は OS X Mavericks v10.9 以降でサポートされています。.NET Core バージョン 2.1 以降では、より新しいバージョンの macOS が必要です。詳細については、https://learn.microsoft.com/ja-jp/dotnet/core/install/windows?tabs=net80&pivots=os-macos を参照してください。

.NET Core 1.0 を使用している場合、.NET Core は OpenSSL を macOS で使用します。この OpenSSL は別個にインストールする必要がある依存関係です。OpenSSL は、バージョン 1.0.1 で TLS 1.2 のサポートを追加し、バージョン 1.1.1 で TLS 1.3 のサポートを追加しました。。

Linux

Linux 上の .NET コアには OpenSSL が必要です。OpenSSL は、多くの Linux ディストリビューションにバンドルされていますが、別個にインストールすることもできます。OpenSSL は、バージョン 1.0.1 で TLS 1.2 のサポートを追加し、バージョン 1.1.1 で TLS 1.3 のサポートを追加しました。最新バージョンの .NET Core (2.1 以降) を使用していて、パッケージマネージャーをインストールしている場合は、通常、最新バージョンの OpenSSL がインストール済みです。

これを調べるには、ターミナルで **`openssl version`** を実行し、バージョンが 1.0.1 より新しいことを確認します。

特定のランタイムライブラリまたは .NET Framework の最小バージョンが必要です。

新しいバージョンの .NET Framework (4.7 以降) と新しいバージョンの Windows (クライアントの場合は Windows 8 以上、サーバーの場合は Windows Server 2012 以降) を実行している場合、TLS 1.2 はデフォルトで有効化され、使用されています。

オペレーティングシステム設定 (.NET Framework 3.5 ~ 4.5.2) を使用しない .NET Framework ランタイムを使用している場合、 AWS SDK for .NET はサポートされているプロトコルに TLS 1.1 および TLS 1.2 のサポートを追加しようとします。.NET Framework 3.5 を使用している場合、これが成功するのは、次のように適切なホットパッチがインストールされている場合のみです。

- Windows 10 バージョン 1511 および Windows Server 2016 – KB3156421
- Windows 8.1 および Windows Server 2012 R2 – KB3154520
- Windows Server 2012 – KB3154519
- Windows 7 SP1 および Server 2008 R2 SP1 – KB3154518

> **Warning**
>
> 2024 年 8 月 15 日、 AWS SDK for .NET は .NET Framework 3.5 のサポートを終了し、最小 .NET Framework バージョンを 4.7.2 に変更しました。詳細については、ブログ記事「Important changes coming for .NET Framework 3.5 and 4.5 targets of the AWS SDK for .NET」を参照してください。

アプリケーションを Windows 7 SP1 または Windows Server 2008 R2 SP1 上の新しい .NET Framework で実行している場合は、レジストリで TLS 1.2 のサポートが有効になっていることを確認する必要があります。詳細については、https://learn.microsoft.com/ja-jp/windows-server/security/tls/tls-registry-settings#tls-12 を参照してください。新しいバージョンの Windows の場合、これはデフォルトで有効になっています。

.NET Framework で TLS を使用するための詳細なベストプラクティスについては、Microsoft の記事 (https://learn.microsoft.com/ja-jp/dotnet/framework/network-programming/tls) を参照してください。

AWS Tools for PowerShell

AWS Tools for PowerShell AWS サービスへのすべての呼び出し AWS SDK for .NET に を使用します。環境の動作は、次のように、実行している Windows PowerShell のバージョンによって異なります。

Windows PowerShell 2.0 ~ 5.x

Windows PowerShell 2.0 ~ 5.x は .NET Framework で実行します。どの .NET ランタイム (2.0 または 4.0) が PowerShell で使用されているかは、次のコマンドで確認できます。

```
$PSVersionTable.CLRVersion
```

- .NET ランタイム 2.0 を使用している場合は、 AWS SDK for .NET および .NET Framework 3.5 に関する前述の手順に従います。

> **Warning**
>
> 2024 年 8 月 15 日、 AWS SDK for .NET は .NET Framework 3.5 のサポートを終了し、最小 .NET Framework バージョンを 4.7.2 に変更しました。詳細については、ブログ記

事「Important changes coming for .NET Framework 3.5 and 4.5 targets of the AWS SDK for .NET」を参照してください。

- .NET ランタイム 4.0 を使用している場合は、 AWS SDK for .NET および .NET Framework 4+ に関する前述の手順に従います。

Windows PowerShell 6.0

Windows PowerShell 6.0 以降は、.NET Core で実行します。どのバージョンの .NET Core が使用されているかは、次のコマンドで確認できます。

```
[System.Reflection.Assembly]::GetEntryAssembly().GetCustomAttributes([System.Runtime.Versioning
 $true).FrameworkName
```

AWS SDK for .NET および .NET Core の関連バージョンについては、前述の手順に従ってください。

Xamarin

Xamarin については、https://learn.microsoft.com/ja-jp/xamarin/cross-platform/app-fundamentals/transport-layer-security の手順を参照してください。要約は、以下のとおりです。

Android の場合

- Android 5.0 以降が必要です。
- [プロジェクトのプロパティ]、[Android のオプション]: [HttpClient 実装] を [Android] に設定し、[SSL/TLS 実装] を [Native TLS 1.2+] に設定する必要があります。

iOS の場合

- iOS 7 以降が必要です。
- [プロジェクトのプロパティ]、[iOS ビルド]: [HttpClient 実装] を [NSUrlSession] に設定する必要があります。

macOS の場合

- macOS 10.9 以降が必要です。

- [プロジェクトのオプション]、[ビルド]、[Mac ビルド]: [HttpClient 実装] を [NSUrlSession] に設定する必要があります。

Unity

Unity 2018.2 以降を使用し、.NET 4.x Equivalent スクリプティングランタイムを使用する必要があります。これを設定するには、[Project Settings]、[Configuration]、[Player] の順に選択します。詳細については、https://docs.unity3d.com/2019.1/Documentation/Manual/ScriptingRuntimeUpgrade.html を参照してください。.NET 4.x Equivalent スクリプティングランタイムは、Mono または IL2CPP を実行するすべての Unity プラットフォームで TLS 1.2 のサポートを有効にします。

ブラウザ (Blazor WebAssembly 用)

WebAssembly は、サーバー上ではなくブラウザで実行され、ブラウザを使用して HTTP トラフィックを処理します。したがって、TLS のサポートはブラウザのサポートによって決まります。

Blazor WebAssembly は、ASP.NET Core 3.1 のプレビューで、https://learn.microsoft.com/en-us/aspnet/core/blazor/supported-platforms で説明されているように、WebAssembly をサポートするブラウザでのみサポートされています。すべてのメインストリームブラウザは、WebAssembly をサポートする前に TLS 1.2 をサポートしていました。これに該当するブラウザの場合、アプリを実行すると、アプリは TLS 1.2 経由で通信できます。

詳細と検証については、ブラウザのドキュメントを参照してください。

S3 暗号化クライアントの移行 (V1 から V2)

> **Note**
>
> V2 を使用していて V4 に移行する場合は、「」を参照してくださいS3 暗号化クライアントの移行 (V2 から V4)。

このトピックでは、Amazon Simple Storage Service (S3) 暗号化クライアントのバージョン 1 (V1) からバージョン 2 (V2) にアプリケーションを移行し、移行プロセス全体でアプリケーションの可用性を確保する方法について説明します。

V2 クライアントで暗号化されたオブジェクトは、V1 クライアントでは復号化できません。すべてのオブジェクトを一度に再暗号化する必要をなくして、新しいクライアントへの移行を容易にするため

に、「V1 移行用の」クライアントが提供されています。このクライアントは、V1 と V2 のどちらで暗号化されたオブジェクトでも復号化できますが、オブジェクトを暗号化する際には V1 互換形式のみを使用します。V2 クライアントは、(V1 オブジェクトのサポートが有効になっていれば) V1 と V2 のどちらで暗号化されたオブジェクトでも復号化できますが、オブジェクトの暗号化には V2 互換形式のみを使用します。

移行の概要

この移行は 3 つのフェーズを通じて行われます。各フェーズについては、後ほど詳しく説明します。次のフェーズを開始する前に、共有オブジェクトを使用するすべてのクライアントで各フェーズを完了させる必要があります。

1. 既存のクライアントを V1 移行用クライアントに更新して、新しい形式を読み取るようにします。最初にアプリケーションを更新して、V1 クライアントではなく V1 移行用クライアントに依存するようにします。V1 移行用クライアントを使用すると、新しい V2 クライアントで記述されたオブジェクトと V1 互換形式で記述されたオブジェクトを既存のコードで復号化できます。

   > **Note**
   >
   > V1 移行用クライアントは、移行のみを目的として提供されています。V1 移行用クライアントに移行した後は、V2 クライアントへのアップグレードに進んでください。

2. V1 移行用クライアントを V2 クライアントに移行して、新しい形式を書き込むようにします。次に、アプリケーション内のすべての V1 移行用クライアントを V2 クライアントに置き換え、セキュリティプロファイルを V2AndLegacy に設定します。V2 クライアントでこのセキュリティプロファイルを設定すると、クライアントは V1 互換形式で暗号化されたオブジェクトを復号化できるようになります。
3. V2 クライアントを更新して V1 形式を読み取らないようにします。最後に、すべてのクライアントの V2 への移行が完了し、すべてのオブジェクトが V2 互換フォーマットで暗号化または再暗号化されたら、V2 セキュリティプロファイルを V2AndLegacy の代わりに V2 に設定します。こうすることで、V1 互換形式のオブジェクトが復号化されなくなります。

既存のクライアントを V1 移行用クライアントに更新して、新しい形式を読み取るようにする

V2 暗号化クライアントは、古いバージョンのクライアントではサポートされていない暗号化アルゴリズムを使用します。移行の最初のステップは、V1 復号化クライアントを更新して、新しい形式を読み取れるようにすることです。

V1 移行用クライアントを使用すると、V1 と V2 のどちらで暗号化されたオブジェクトでもアプリケーションで復号化できるようになります。このクライアントは、Amazon.Extensions.S3.Encryption NuGet パッケージの一部です。V1 移行用クライアントを使用するには、各アプリケーションで次の手順を実行します。

1. Amazon.Extensions.S3.Encryption パッケージへの新しい依存関係を構築します。プロジェクトが AWSSDK.S3 または AWSSDK.KeyManagementService パッケージに直接依存している場合は、これらの依存関係を更新または削除して、更新されたバージョンがこの新しいパッケージに取り込まれるようにする必要があります。
2. 該当する `using` ステートメントを、以下のように `Amazon.S3.Encryption` から `Amazon.Extensions.S3.Encryption` に変更します。

   ```
   // using Amazon.S3.Encryption;
     using Amazon.Extensions.S3.Encryption;
   ```

3. アプリケーションを再構築して再デプロイします。

V1 移行用クライアントは V1 クライアントと完全に API 互換であるため、他のコード変更は必要ありません。

V1 移行用クライアントを V2 クライアントに移行して、新しい形式を書き込むようにする

V2 クライアントは、Amazon.Extensions.S3.Encryption NuGet パッケージの一部です。このクライアントを使用すると、V1 と V2 のどちらで暗号化されたオブジェクトでもアプリケーションで復号化できるようになりますが (そのように設定された場合)、オブジェクトの暗号化には V2 互換形式のみを使用します。

既存のクライアントを更新して新しい暗号化形式を読み取るようにした後で、アプリケーションが V2 暗号化および復号化クライアントを使用するように安全に更新できます。V2 クライアントを使用するには、各アプリケーションで次の手順を実行します。

1. EncryptionMaterials を EncryptionMaterialsV2 に変更します。

   a. KMS を使用する場合:

      i. KMS キー ID を指定します。

      ii. 使用している暗号化方法、すなわち KmsType.KmsContext を宣言します。

      iii. このデータキーに関連付ける暗号化コンテキストを KMS に対して指定します。空の辞書を送信することはできますが (Amazon 暗号化コンテキストは引き続きマージされます)、追加のコンテキストを指定することをお勧めします。

   b. ユーザー指定のキーラップ方式を使用する場合 (対称暗号化または非対称暗号化):

      i. 暗号化マテリアルを含む AES または RSA インスタンスを指定します。

      ii. 使用する暗号化アルゴリズム、すなわち SymmetricAlgorithmType.AesGcm または AsymmetricAlgorithmType.RsaOaepSha1 を宣言します。

2. AmazonS3CryptoConfiguration を AmazonS3CryptoConfigurationV2 に変更し、SecurityProfile プロパティを SecurityProfile.V2AndLegacy に設定します。

3. AmazonS3EncryptionClient を AmazonS3EncryptionClientV2 に変更します。このクライアントは、前のステップで新たに変換された AmazonS3CryptoConfigurationV2 および EncryptionMaterialsV2 オブジェクトを受け取ります。

例: KMS から KMS+コンテキスト

移行前

```
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);
```

移行後

```
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);
```

例: 対称アルゴリズム (AES-CBC から AES-GCM キーラップ)

StorageMode は ObjectMetadata または InstructionFile のいずれかになります。

移行前

```
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);
```

移行後

```
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**
>
> AES-GCM で復号化する場合は、復号化されたデータの使用を開始する前に、オブジェクト全体を最後まで読み取ってください。これは、オブジェクトが暗号化されていた時点から変更されていないことを確認するためのステップです。

例: 非対称アルゴリズム (RSA から RSA-OAEP-SHA1 キーラップ)

`StorageMode` は `ObjectMetadata` または `InstructionFile` のいずれかになります。

移行前

```
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);
```

移行後

```
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);
```

V2 クライアントを更新して V1 形式を読み取らないようにする

最終的に、すべてのオブジェクトが V2 クライアントを使用して暗号化または再暗号化されます。この変換が完了した後、次のスニペットに示すように `SecurityProfile` プロパティを `SecurityProfile.V2` として、V2 クライアントの V1 互換性を無効にできます。

```
//var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy);
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2);
```

S3 暗号化クライアントの移行 (V2 から V4)

> **Note**
>
> V1 を使用していて V4 に移行する場合は、まず V2 に移行する必要があります。「S3 暗号化クライアントの移行 (V1 から V2)」を参照してください。

このトピックでは、Amazon Simple Storage Service (S3) 暗号化クライアントのバージョン 2 (V2) からバージョン 4 (V4) にアプリケーションを移行し、移行プロセス全体でアプリケーションの可用性を確保する方法について説明します。V4 は、AES-GCM をコンテンツ暗号化のキーコミットメントとして使用し、コミットメントポリシーを導入して、キー置換攻撃に対するセキュリティを強化します。

V4 クライアントは、Amazon.Extensions.S3.Encryption NuGet パッケージで使用できます。

> **Important**
>
> 重大な変更: `RequireEncryptRequireDecrypt`ポリシーで設定された V4 クライアントはV1 または V2 クライアントで暗号化されたオブジェクトを復号できません。キーコミットメントを使用して VV4-encrypted V2 クライアントのみです。`RequireEncryptRequireDecrypt` ポリシーに移行する前に、キーコミットメントが有効になっている V4 クライアントを使用して既存のデータをすべて再暗号化する必要があります。

V4 の概念について

V4 は、AES-GCM をコンテンツ暗号化のキーコミットメントとして使用し、暗号化されたデータの保護を強化する主要なセキュリティ概念を紹介します。

コミットメントポリシー

コミットメントポリシーは、暗号化および復号オペレーション中に暗号化クライアントがキーコミットメントを処理する方法を制御します。V4 は 3 つのコミットメントポリシーをサポートしています。

`ForbidEncryptAllowDecrypt`

暗号化: コミットメントなし

復号: コミットされていないオブジェクトを許可する

セキュリティ: コミットメントを強制せず、改ざんを許可する場合があります

互換性: すべての V2 および V4 実装は、このポリシーで暗号化されたオブジェクトを読み取ることができます

`RequireEncryptAllowDecrypt`

暗号化: キーコミットメントあり

復号: コミットするオブジェクトとコミットしないオブジェクトの両方を許可する

セキュリティ: 新しいオブジェクトはキー置換攻撃から保護され、古いオブジェクトは引き続き読み取り可能

互換性: V4 のみがこのポリシーをサポート

`RequireEncryptRequireDecrypt`(V4 のデフォルト)

暗号化: キーコミットメントあり

復号: オブジェクトのみをコミットする

セキュリティ: セキュリティを最大化するための完全コミットメントの適用

互換性: V4 のみがこのポリシーをサポート

キーコミットメントを持つ AES GCM

V4 は、AES-GCM をコンテンツ暗号化のキーコミットメントとして使用し、セキュリティを強化します。

- 改ざん保護: 暗号化されたデータにキーを暗号化バインドすることで、キー置換攻撃から保護します。
- バージョンの互換性: キーコミットメントで暗号化されたオブジェクトは、V4 クライアント以降のバージョンでのみ復号できます。

> **Warning**
>
> 本番環境でキーコミットメント暗号化を有効にする前に、オブジェクトを復号する必要があるすべてのアプリケーションが、V2 クライアントが非推奨になったときに V4 以降にアップグレードされていることを確認してください。 V2

既存のクライアントを更新して V4 形式を読み取る

V4 暗号化クライアントは、古いバージョンのクライアントがサポートしていない暗号化アルゴリズムを使用します。移行の最初のステップは、新しい V4 形式を読み取れるように V2 クライアントを更新することです。 V4

NuGet パッケージの依存関係を更新する

V4 サポートを含む最新バージョンの Amazon.Extensions.S3.Encryption NuGet パッケージを使用するようにアプリケーションを更新します。各アプリケーションで次の手順を実行します。

1. 最新の Amazon.Extensions.S3.Encryption パッケージに更新します。プロジェクトが AWSSDK.S3 または AWSSDK.KeyManagementService パッケージに直接依存している場合は、これらの依存関係を更新または削除して、更新されたバージョンがこの新しいパッケージに取り込まれるようにする必要があります。
2. `using` ステートメントが正しい名前空間を参照していることを確認します。

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;
```

3. アプリケーションを再構築して再デプロイします。

既存の V2 クライアントは、更新されたパッケージで引き続き動作し、(使用されるコミットメントポリシーに応じて) V4 クライアントによって暗号化されたオブジェクトを復号できます。

アプリケーションの構築とデプロイ

NuGet パッケージの依存関係を更新した後:

1. すべての依存関係が正しく解決されるようにアプリケーションを構築します。
2. 開発環境でアプリケーションをテストして、既存の機能が引き続き機能することを確認します。
3. 更新されたアプリケーションを本番環境にデプロイします。

この更新により、既存の V2 クライアントは V4 クライアントによって暗号化されるオブジェクトを復号できるため、移行プロセス中の互換性が確保されます。

V4 クライアントへの移行

新しい暗号化形式を読み取るように既存のクライアントを更新した後、V4 暗号化および復号クライアントを使用するようにアプリケーションを安全に更新できます。V4 クライアントは、既存の暗号化されたオブジェクトとの互換性を維持しながら、キーコミットメントを通じてセキュリティを強化します。

4-Step移行プロセス

V2 から V4 への移行は、互換性とセキュリティを確保するための構造化された 4 ステップのプロセスに従います。各ステップは、次のステップに進む前にすべてのアプリケーションにデプロイする必要がある特定の設定を表します。

1. ステップ 0: V2 クライアント (開始点) - 既存の V2 実装
2. ステップ 1: V2 互換の V4 - V2 V2-compatible 互換の暗号化動作を維持しながら V4 クライアントに移行する
3. ステップ 2: キーコミットメント書き込み付き V4 - レガシーオブジェクトの復号化を許可しながら、キーコミットメントによる暗号化を開始する
4. ステップ 3: フルエンフォースメントを使用する V4 - 暗号化と復号の両方にキーコミットメントが必要です

ステップ 0: V2 クライアント (開始点)

これは、既存の V2 クライアント設定を表します。このステップでは、移行前の開始状態を示します。

```
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
});
```

ステップ 1: V2 互換の V4 V2

V2 と同じ動作を維持しながらV2V4 クライアントに移行します。このステップでは、`ForbidEncryptAllowDecrypt`ポリシーを使用してコミットメントなしで暗号化し、すべてのオブジェクトの復号化を許可します。

```
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
});
```

動作: コミットメントなしで暗号化し、コミットしているオブジェクトとコミットしていないオブジェクトの両方を復号できます。V2 の動作と同じです。

ステップ 2: キーコミットメント書き込みを含む V4

復号の下位互換性を維持しながら、キーコミットメントによる暗号化を開始します。このステップでは、 `RequireEncryptAllowDecrypt`ポリシーを使用します。

> **⚠ Warning**
>
> ステップ 2 をデプロイする前に、キーコミットメントの暗号化を処理するために、すべてのリーダーがステップ 1 以降に更新されていることを確認します。

```
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
});
```

動作: コミットメントで暗号化し、コミットしているオブジェクトとコミットしていないオブジェクトの両方を復号できます。新しいオブジェクトは、キー置換攻撃から保護されます。

ステップ 3: フルエンフォースメントを使用する V4

暗号化と復号の両方にキーコミットメントが必要です。このステップでは、セキュリティを最大化するために RequireEncryptRequireDecryptポリシーを使用します。

> **⚠ Warning**
>
> ステップ 3 をデプロイする前に、システム内のすべてのオブジェクトがキーコミットメントで再暗号化されていることを確認します (ステップ 2)。このステップでは、コミットメントなしで暗号化されたオブジェクトを復号化できません。

```
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
});
```

動作: コミットメントで暗号化し、コミットメントで暗号化されたオブジェクトのみを復号します。キー置換攻撃に対する最大セキュリティ。

その他の設定例

このセクションでは、移行中にさまざまなオプションを使用して V4 クライアントを設定するための追加の例を示します。

レガシーサポートの有効化

V4 クライアントが V1 および V2 クライアントによって暗号化されたオブジェクトを読み取れるようにするには、レガシー復号を許可するコミットメントポリシーでクライアントを設定します。

```
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
```

新しいオブジェクトのセキュリティを強化して暗号化しながら、古いクライアントによって暗号化されたオブジェクトを復号する必要がある場合は、移行時にこの設定を使用します。

ストレージ方法の設定

V4 は、暗号化メタデータの 2 つのストレージ方法をサポートしています。ユースケースに最適な方法を選択します。

オブジェクトメタデータ (デフォルト)

```
var configuration = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.RequireEncryptAllowDecrypt)
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
// Encryption metadata is stored in S3 object metadata
```

指示ファイル

```
var configuration = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.RequireEncryptAllowDecrypt)
{
    StorageMode = CryptoStorageMode.InstructionFile
};
// Encryption metadata is stored in a separate S3 object (instruction file)
```

他の目的でオブジェクトメタデータを保持InstructionFileする必要がある場合や、メタデータサイズに制限があるオブジェクトを操作する場合は、 を使用します。

コミットメントポリシーの設定

セキュリティ要件と移行フェーズに基づいて、適切なコミットメントポリシーを選択します。

移行フェーズ (V2 互換性)

```
// For migration: encrypt without commitment, allow all decryption
var migrationConfig = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.ForbidEncryptAllowDecrypt);
```

移行フェーズ (推奨)

```
// For transition: encrypt with commitment, allow legacy decryption
var transitionConfig = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.RequireEncryptAllowDecrypt);
```

完全なセキュリティフェーズ

```
// For maximum security: require commitment for both encryption and decryption
var secureConfig = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.RequireEncryptRequireDecrypt);
```

最初の移行ForbidEncryptAllowDecrypt中に から開始し、移行フェーズRequireEncryptAllowDecryptのために に移動し、最後にすべてのクライアントがアップグレードされ、すべてのオブジェクトがコミットメントで再暗号化されたRequireEncryptRequireDecryptときに を使用します。

に関する特別な考慮事項 AWS SDK for .NET

このセクションでは、通常の設定または手順が適切でないか、十分でない特殊なケースに関する考慮事項について説明します。

トピック

- のアセンブリの取得 AWS SDK for .NET
- アプリケーションでの認証情報とプロファイルへのアクセス
- Unity のサポートに関する特別な考慮事項
- Xamarin のサポートに関する特別な考慮事項

のアセンブリの取得 AWS SDK for .NET

このトピックでは、プロジェクトに使用するために、 AWSSDK アセンブリを入手して、ローカル (またはオンプレミス) に保存する方法について説明します。この方法は、SDK リファレンスの処理には推奨されませんが、一部の環境では必須です。

> **Note**
>
> SDK リファレンスを処理するには、各プロジェクトで必要とされる NuGet パッケージのみをダウンロードしてインストールする方法が推奨されます。その方法については、「NuGet を使用して AWSSDK パッケージをインストールする」を参照してください。

プロジェクトごとに NuGet パッケージをダウンロードしてインストールできないか、許可されていない場合は、次のオプションを使用します。

ZIP ファイルをダウンロードして抽出する

（これは への参照を処理するための推奨方法ではないことに注意してください） AWS SDK for .NET。

1. 次の ZIP ファイルのいずれかをダウンロードします。

   - aws-sdk-net8.0.zip – .NET 8 以降をサポートするアセンブリ。
   - aws-sdk-netcoreapp3.1.zip-.NET Core 3.1 以降をサポートするアセンブリ。

- aws-sdk-netstandard2.0.zip-.NET 標準 2.0 と 2.1 をサポートするアセンブリ。
- aws-sdk-net472.zip - .NET Framework 4.5 以降をサポートするアセンブリ。

2. アセンブリをファイルシステム上の「ダウンロード」フォルダなどに展開します。場所は重要ではありません。このフォルダを書き留めます。
3. プロジェクトをセットアップしたら、「NuGet を使用せずに AWSSDK アセンブリをインストールする」の説明に従って、このフォルダから必要なアセンブリを取得します。

アプリケーションでの認証情報とプロファイルへのアクセス

認証情報を使用するための推奨方法は、「」で説明されているように、 AWS SDK for .NET が認証情報を検索して取得できるようにすることです認証情報とプロファイルの解決。

ただし、プロファイルと認証情報をアクティブに取得し、 AWS サービスクライアントの作成時にそれらの認証情報を明示的に使用するようにアプリケーションを設定することもできます。

プロファイルと認証情報を能動的に取得するには、Amazon.Runtime.CredentialManagement 名前空間からクラスを使用します。

- AWS 認証情報ファイル形式 (デフォルトの場所にある共有 AWS 認証情報ファイルまたはカスタム認証情報ファイル) を使用するファイル内のプロファイルを検索するには、SharedCredentialsFile クラスを使用します。簡潔にするために、本テキストではこの形式のファイルを単に認証情報ファイルと呼ぶ場合があります。
- SDK ストアでプロファイルを検索するには、NetSDKCredentialsFile クラスを使用します。
- 認証情報ファイルと SDK ストアの両方で検索するには、クラスプロパティの設定に応じて CredentialProfileStoreChain クラスを使用します。

  このクラスを使用して、プロファイルを検索できます。また、このクラスを使用して、`AWSCredentialsFactory` クラス (次に説明) を使用する代わりに AWS 、認証情報を直接リクエストすることもできます。
- プロファイルからさまざまなタイプの認証情報を取得または作成するには、AWSCredentialsFactory クラスを使用します。

以下のセクションでは、これらのクラスの例を示します。

クラス CredentialProfileStoreChain の例

TryGetAWSCredentials または TryGetProfile メソッドを使用して、CredentialProfileStoreChain クラスから認証情報またはプロファイルを取得できます。クラスの ProfilesLocation プロパティにより、次のようにメソッドの動作が決まります。

- ProfilesLocation が null または空の場合、プラットフォームがサポートしている場合は SDK ストアを検索し、デフォルトの場所で共有 AWS 認証情報ファイルを検索します。
- ProfilesLocation プロパティに値が含まれている場合は、プロパティで指定された認証情報ファイルを検索します。

SDK ストアまたは共有認証情報ファイルから AWS 認証情報を取得する

この例では、CredentialProfileStoreChain クラスを使用して認証情報を取得した後、その認証情報を使用して AmazonS3Client オブジェクトを作成する方法を示しています。認証情報は、SDK ストアまたはデフォルトの場所にある共有 AWS 認証情報ファイルから取得できます。

この例では、Amazon.Runtime.AWSCredentials クラスも使用します。

```
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}");
    }
}
```

SDK ストアまたは共有 AWS 認証情報ファイルからプロファイルを取得する

この例では、CredentialProfileStoreChain クラスを使用してプロファイルを取得する方法を示しています。認証情報は SDK ストアから取得することも、デフォルトの場所にある共有 AWS 認証情報ファイルから取得することもできます。

この例では、CredentialProfile クラスも使用します。

```
var chain = new CredentialProfileStoreChain();
```

```
CredentialProfile basicProfile;
if (chain.TryGetProfile("basic_profile", out basicProfile))
{
    // Use basicProfile
}
```

カスタム認証情報ファイルからの認証情報の取得

この例では、CredentialProfileStoreChain クラスを使用して認証情報を取得する方法を示しています。認証情報は、 AWS 認証情報ファイル形式を使用するが、別の場所にあるファイルから取得されます。

この例では、Amazon.Runtime.AWSCredentials クラスも使用します。

```
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
}
```

クラス SharedCredentialsFile と AWSCredentialsFactory の例

SharedCredentialsFile クラスを使用した AmazonS3Client の作成

この例では、共有 AWS 認証情報ファイルでプロファイルを検索し、プロファイルから認証情報を作成し AWS 、その認証情報を使用して AmazonS3Client オブジェクトを作成する方法を示します。この例では、SharedCredentialsFile クラスを使用します。

この例では、CredentialProfile クラスと Amazon.Runtime.AWSCredentials クラスも使用します。

```
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**
>
> NetSDKCredentialsFile クラスは、SharedCredentialsFile オブジェクトの代わりに新しい NetSDKCredentialsFile オブジェクトをインスタンス化する点を除けば、まったく同じ方法で使用できます。

Unity のサポートに関する特別な考慮事項

Unity アプリケーションに AWS SDK for .NET および .NET Standard 2.0 を使用する場合、アプリケーションは NuGet を使用するのではなく、アセンブリ (DLL ファイル) を直接参照 AWS SDK for .NET する必要があります。この要件を考慮すると、実行する必要がある重要なアクションは次のとおりです。

- AWS SDK for .NET アセンブリを取得し、プロジェクトに適用する必要があります。これを行う方法については、トピック「AWSSDK アセンブリを入手する」の「ZIP ファイルをダウンロードして抽出する」を参照してください。

- Unity プロジェクトには、AWSSDK.Core の DLLs および使用しているその他の AWS サービスとともに、次の DLLs を含める必要があります。 AWSSDK のバージョン 3.5.109 以降 AWS SDK for .NET、.NET Standard ZIP ファイルにはこれらの追加の DLLsが含まれています。
  - Microsoft.Bcl.AsyncInterfaces.dll
  - system.runtime.CompilerServices.unsafe.dll
  - system.tasks.Extensions.dll

- IL2CPP を使用して Unity プロジェクトをビルドする場合、コードのストリッピングが行われないようにするため、`link.xml` ファイルをアセットフォルダに追加する必要があります。`link.xml` ファイルには、使用しているすべての AWSSDK アセンブリがリストされている必要があり、各アセンブリには `preserve="all"` 属性が含まれている必要があります。以下のスニペットは、このファイルの例です。

```
<linker>
    <assembly fullname="AWSSDK.Core" preserve="all"/>
    <assembly fullname="AWSSDK.DynamoDBv2" preserve="all"/>
    <assembly fullname="AWSSDK.Lambda" preserve="all"/>
</linker>
```

> **Note**
>
> この要件に関連する興味深い背景情報については、https://aws.amazon.com/blogs/developer/referencing-the-aws-sdk-for-net-standard-2-0-from-unity-xamarin-or-uwp/ の記事を参照してください。

Xamarin のサポートに関する特別な考慮事項

Xamarin プロジェクト (新規および既存) は、.NET Standard 2.0 を対象とする必要があります。「Xamarin.Forms での .NET Standard 2.0 のサポート」および「.NET 実装サポート」を参照してください。

「ポータブルクラスライブラリと Xamarin」の情報も参照してください。

の API リファレンス AWS SDK for .NET

AWS SDK for .NET には、 AWS サービスへのアクセスに使用できる API が用意されています。API で使用できるクラスとメソッドを確認するには、AWS SDK for .NET API リファレンスを参照してください。

上記の全般的なリファレンスに加えて、ガイド付きコードの例 セクションの以下の各例には、その例で使用されている特定のメソッドとクラスのリファレンスが含まれています。

API リファレンスバージョンについて

前述の API リファレンスは、 のバージョン 3.0 以降用です AWS SDK for .NET。

古いバージョンの SDK からの移行については、「」を参照してください。 プロジェクトを移行する

SDK API リファレンスの以前のバージョンで廃止されたコンテンツを見つけるには、次の項目 (複数可) を参照してください。

- AWS SDK for .NET API リファレンス V1 (廃止)
- AWS SDK for .NET API リファレンス V2 (廃止)

廃止された AWS SDK for .NET API リファレンスを表示するには、それを抽出してウェブブラウザを設定する必要があります。次に示す手順は、これを行う方法の例です。これらは、Windows システムで Google Chrome ウェブブラウザを使用することに基づいています。特定のウェブブラウザとオペレーティングシステムに適応させます。

廃止された API リファレンス V1 を表示する

1. 廃止された AWS SDK for .NET API リファレンス V1 の ZIP ファイルをダウンロードします。デフォルトでは、ZIP ファイルの名前は ですsdkfornet-api-ref_v1_deprecated.zip。この名前は、これらの手順全体で使用されます。
2. ZIP ファイルを任意のフォルダに配置します。これらの手順では、フォルダ名は と見なされますC:\work\temp\api-refs\V1。
3. ZIP ファイルを右クリックし、すべて抽出を選択します。これらの手順C:\work\temp\api-refs\V1\sdkfornet-api-ref_v1用のデフォルトの場所を受け入れます。

4. C:\work\temp\api-refs\V1 フォルダに Google Chrome のショートカットを作成します。元のアプリケーションを移動しないように注意してください。
5. 新しいショートカットのプロパティで、次のフィールドを設定します。

   - ターゲット: "*<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"

     --user-data-dir 引数には、環境に適したフォルダ名を使用します。フォルダが存在する必要はありません。

   - 開始場所: C:\work\temp\api-refs\V1
6. ショートカットに適切な名前を付けます。
7. ショートカットを開いて、古い API リファレンスを表示します。

廃止された API リファレンス V2 を表示する

1. 廃止された AWS SDK for .NET API リファレンス V2 の ZIP ファイルをダウンロードします。デフォルトでは、ZIP ファイルの名前は ですsdkfornet-api-ref_v2_deprecated.zip。この名前は、これらの手順全体で使用されます。
2. ZIP ファイルを任意のフォルダに配置します。これらの手順では、フォルダ名は と見なされますC:\work\temp\api-refs\V2。
3. ZIP ファイルを右クリックし、すべて抽出を選択します。これらの手順C:\work\temp\api-refs\V2\sdkfornet-api-ref_v2用のデフォルトの場所を受け入れます。
4. C:\work\temp\api-refs\V2 フォルダに Google Chrome のショートカットを作成します。元のアプリケーションを移動しないように注意してください。
5. 新しいショートカットのプロパティで、次のフィールドを設定します。

   - ターゲット: "*<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"

     --user-data-dir 引数には、環境に適したフォルダ名を使用します。フォルダが存在する必要はありません。

   - 開始場所: C:\work\temp\api-refs\V2
6. ショートカットに適切な名前を付けます。

7. ショートカットを開いて、古い API リファレンスを表示します。

ドキュメント履歴

次の表は、AWS SDK for .NET デベロッパーガイドの前回リリースからの重要な変更点をまとめたものです。このドキュメントの更新に関する通知については、RSS フィードを購読してください。

変更	説明	日付
最新情報	.NET CLI 用の AWS デプロイツールのバージョン 2.0 に関する注記を追加しました。	2025 年 10 月 17 日
AWS OpsWorks	AWS OpsWorks はサポート終了に達しました。ドキュメントから削除されました。	2025 年 9 月 16 日
最新情報	の V3 End-of-supportが発表されました AWS SDK for .NET。	2025 年 8 月 21 日
最新情報	分散キャッシュプロバイダーの一般提供	2025 年 7 月 2 日
メッセージ処理フレームワーク	AWS Message Processing Framework for .NET が一般公開されました。	2025 年 5 月 5 日
最新情報	のバージョン 4 AWS SDK for .NET は一般利用可能です。アプリケーションを V4 に移行する方法については、「バージョン 4 への移行」を参照してください。	2025 年 4 月 28 日
のバージョン 4 への移行 AWS SDK for .NET	を使用して構築されたアプリケーションのバージョン 4 AWS SDK for .NET への移行に関するプレビュー情報を追加しました。	2025 年 3 月 3 日

での .NET Aspire AWS との統合 AWS SDK for .NET	AWS が .NET ASPIRE とどのように統合されているかに関する情報を に含めました AWS SDK for .NET。	2025 年 2 月 15 日
可観測性	オブザーバビリティに関する GA リリースを発表しました。	2025 年 2 月 10 日
最新情報	整合性保護のための新しいデフォルト動作に関する情報を追加しました。	2025 年 1 月 15 日
S3 暗号化クライアント V4 の移行	V2 から V4 への移行に関する情報を追加しました。	2024 年 12 月 9 日
最新情報	AWS SDK for .NET バージョン 4 の 4 番目のプレビューリリースに関する情報を追加しました。	2024 年 11 月 15 日