開発者ガイド

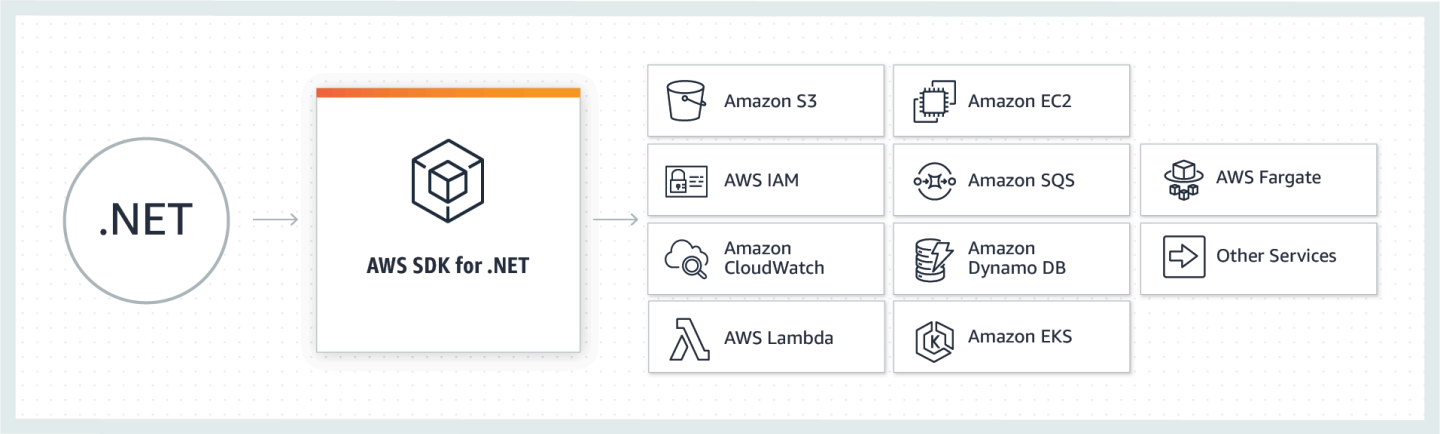
AWS SDK for .NET

AWS SDK for .NET: 開発者ガイド

Table of Contents

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) などの費用対効果、スケーラビリティ、信頼性に優れた AWS サービスを活用する.NET アプリケーションを簡単に構築できます。SDK は、.NET デベロッパーにとって一貫性のある使い慣れたライブラリセットを提供することで、 サービスの使用 AWS を簡素化します。



(わかりました。 クイックツアー を設定して実行する準備ができました)。 ???

このバージョンについて

> **Note**
>
> このドキュメントは、 のバージョン 3.0 以降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。
> 移行の詳細については、「プロジェクトを移行する」を参照してください。
> 以前のバージョンの の非推奨コンテンツを検索するには AWS SDK for .NET、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

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

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

- 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 SDK for .NET に関連する AWS ツール
- AWS SDK とツールのリファレンスガイド
- その他のリソース

AWS SDK for .NET に関連する AWS ツール

Tools for Windows PowerShell と Tools for PowerShell Core

AWS Tools for Windows PowerShell と AWS Tools for PowerShell Core は、AWS SDK for .NET が公開している機能に基づいて構築された PowerShell モジュールです。AWS PowerShell ツールを使用すると、PowerShell プロンプトから AWS リソースへの操作を実行するスクリプトを作成できます。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、これらのアプリケーションを AWS CloudFormation スタックにデプロイする。
- 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する。

Toolkit for Visual Studio

AWS Toolkit for Visual Studio は、Amazon Web Services を使用した .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

AWS Toolkit for Microsoft Azure DevOps は、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で AWS のサービスを簡単に使用するためのタスクを追加します。Amazon S3、AWS Elastic Beanstalk、AWS CodeDeploy、Lambda、AWS CloudFormation、Amazon Simple Queue Service (Amazon SQS)、および Amazon SNS を使用できます。Windows PowerShell モジュールと AWS Command Line Interface (AWS CLI) を使用してコマンドを実行することもできます。

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

AWS SDK とツールのリファレンスガイド

「AWS SDK およびツール リファレンス ガイド」には、多くの AWS SDK、ツールキット、および AWS CLI に関連し重要な情報が含まれています。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、ロールの設定
- 設定と認証設定のリファレンス
- AWS Common Runtime (CRT) ライブラリ
- AWS SDK とツールのメンテナンスポリシー
- AWS SDK とツールのバージョンサポートマトリクス

その他のリソース

サポートされる サービス

AWS SDK for .NET は、AWS インフラストラクチャ製品のほとんどをサポートしており、サービスの追加も頻繁に行っています。SDK によってサポートされている AWS のサービスの一覧については、SDK README ファイルを参照してください。

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
- の最新情報 AWS SDK for .NET
- ドキュメント履歴

AWS SDK for .NET のホームページ

AWS SDK for .NET の詳細については、SDK のホームページ (https://aws.amazon.com/sdk-for-net/) 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、「AWS SDK for .NET API リファレンス」を参照してください。

AWSre: POST (以前のAWSフォーラム)

AWS に関する質問やフィードバックを提供するには、AWSre: BB Post (特に AWS SDK for .NET のトピック) にアクセスしてください。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。AWS SDK for .NET のトピックをフォローするには、「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 .NET Configuration Extension for Systems Manager
- AWS Extensions .NET Core セットアップ
- AWS Logging .NET
- 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:AWSでの.NET アプリケーションの開発とデプロイ
- AWS Microservice Extractor for .NET
- Porting Assistant for .NET
- SDK とツールのリファレンスガイドAWS

AWS SDK for .NET の開始方法

AWS SDK for .NETを使用するには、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項目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
- 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

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、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) AWS ツールキット (選択したエディタとお使いのオペレーティングシステムで使用できる場合)。

  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) AWS Toolkit for Visual Studio。これは、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で SDK 認証を設定します

AWSで SDK 認証を設定します

AWS のサービス を使用して開発する際には、AWS によりコードがどのように認証するかを設定する必要があります。環境と利用可能な AWS のアクセスに応じて、AWS リソースへのプログラムによるアクセスを設定する方法は異なります。

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しており、雇用主から認証方法を与えられておらず、AWS IAM Identity Center を使って一時的な認証情報を取得する予定であ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. IAM Identity Center の有効化と設定
2. IAM Identity Center を使用するように SDK を構成します。
3. AWS アクセスポータルセッションを開始する

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するために使用する AWS CLI。

- SDK から参照できる構成値のセットを含む[default] プロファイル を含む AWS config 共有ファイル。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 AWS に送信する前に、プロファイルの SSO トークンプロバイダー設定を使用して認証情報を取得します。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**
>
> 認証にAWS IAM Identity Centerを使用している場合は、SSO ソリューションが機能するように、アプリケーションで次の 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

AWS のサービス にアクセスするアプリケーションを実行する前に、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、AWS CLI で次のコマンドを実行します。

```
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。

既にアクティブなセッションが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って `aws sso login` を実行している場合は、認証情報を入力するように要求されません。
> サインインプロセス中に、データへの AWS CLI 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` の名前のさまざまなバリエーションが含まれる場合があります。

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。

AWS SDK for .NETのクイックツアーに参加してください

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 のサービス向けのソフトウェア開発とコード例については、「 AWS サービスの使用」を参照してください。その他のコード例については、「AWS SDK for .NET コード例」を参照してください。

トピック

- AWS SDK for .NET 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
- 次のステップ

AWS SDK for .NET 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

このチュートリアルでは、AWS SDK for .NET と .NET Core を使用して、クロスプラットフォーム開発を行います。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、「ツールチェーンのインストールと設定」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ます。

```
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、*BUCKET-NAME* を、選択したバケットの名前に置き換えます。

```
dotnet run BUCKET-NAME
```

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。

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

このチュートリアルでは、Windows 上の AWS SDK for .NET と Visual Studio および .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、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、「ツールチェーンのインストールと設定」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に」と入力します。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 バージョンは選択しないでください。選択する場合は、.NET Framework 4.6.2 以降を使用してください。

2. 新しく作成されたプロジェクトをロードしたら、ツール、NuGetパッケージマネージャー、ソリューションの NuGet パッケージの管理を選択します。
3. 次の NuGet パッケージを参照してプロジェクトにインストールします: `AWSSDK.S3`、、`AWSSDK.SecurityTokenAWSSDK.SSO`、および `AWSSDK.SSOOIDC`

   このプロセスでは、 NuGet パッケージNuGet マネージャー から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 パッケージが正確にわかっているため、このステップはここで実行できます。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。ベースとしてdotnet-quicktour-s「3-1-winvs-」を使用し、GUID や名前など、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 *BUCKET-NAME* を、選択したバケットの名前に置き換えます。

   ```
   S3CreateAndList BUCKET-NAME
   ```

   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 Properties、デバッグを選択し、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、AWS のリソースまたは開発環境内のリソース (ファイルやフォルダなど) である可能性があります。

AWS SDK for .NET のツアーが終わったので、プロジェクトを始めたいとお考えかもしれません。

新しいプロジェクトを開始する

新しいプロジェクトを開始して AWS のサービスにアクセスするには、いくつかのテクニックを使用できます。これらのテクニックの一部を次に示します。

- AWS で.NET 開発を初めて行う場合や、AWS SDK for .NET を初めて使用する場合は、クイックツアーをするで完全な例をご確認ください。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。

  これを確認する方法については、AWS Toolkit for Visual Studio ユーザーガイドの「AWS サービスでの作業」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。

- AWS Toolkit for Visual Studio を使用せずに Visual Studio を使用して Windows で開発する場合、一般的なテクニックを使用して新しいプロジェクトを作成します。

  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 での作業方法の例については、「ガイダンス付きのコード例」を参照してください。

> **Important**
>
> AWS IAM Identity Centerを認証にを使用している場合は、SSO 解決が機能するように SSO 解決が機能するように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。

AWS リージョンを設定する

AWS リージョンにより、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することができます。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、AWS 全般のリファレンスの「DD 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。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アクセスする AWS サービスクライアントを作成できます。また、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を使用して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

アプリケーションのいずれかの AWS サービスクライアント用に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

アプリケーションによって作成されるすべての AWS サービスクライアント用にリージョンを指定する方法は複数あります。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。

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

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

AWS リージョン の指定に上記のいずれの方法も使用されない場合、AWS SDK for .NET は AWS サービス クライアントが動作するリージョンを見つけようとします。

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. 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]`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投げます。

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) リージョンでは使用できません。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 リージョンでは使用できません。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 サービスに関する新しい特別な情報

新しい AWS のサービスは、最初にいくつかのリージョンで開始された後で、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。

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 アセンブリの取得」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
   ```

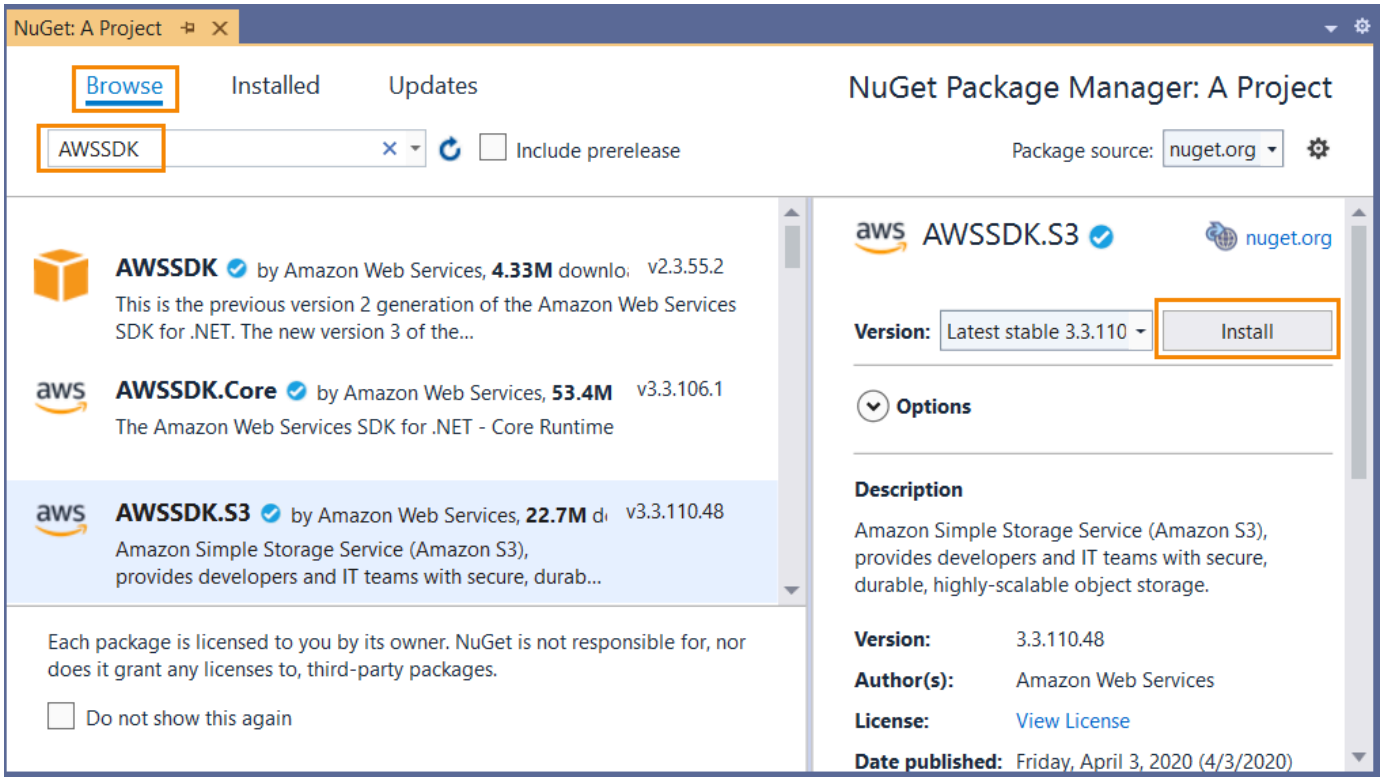
3. プロジェクトのディレクトリで、その .NET CLI コマンドを実行します。NuGet は、AWSSDK.Core などの依存関係があるものもすべてインストールします。

> **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 アセンブリの取得」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。

認証情報とプロファイルの解決

AWS SDK for .NET は定められた順序で認証情報を検索し、最初に利用できるセットを現在のアプリケーションで使用します。

認証情報の検索順序

1. 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 での記載に従い、AWS サービスクライアントで明示的に設定されている認証情報。

> **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. 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。

3. 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。
4. [default] 認証情報プロファイル。
5. AWS_ACCESS_KEY_ID、AWS_SECRET_ACCESS_KEY および AWS_SESSION_TOKEN 環境変数がすべて空でない場合に変数を使用して生成される SessionAWSCredentials。
6. AWS_ACCESS_KEY_ID、AWS_SECRET_ACCESS_KEY 環境変数が両方とも空でない場合に変数を使用して作成される BasicAWSCredentials。
7. Amazon ECS タスクでのタスク用 IAM ロール
8. 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。それ以外の場合 (プレリリーステストなど) は、ウェブアプリケーションがサーバー上でアクセス可能な、AWS 認証情報ファイル形式を使用したファイルに認証情報を保存します。

プロファイルの解決

2 つの異なる認証情報ストレージメカニズムがあるため、それを使用するように AWS SDK for .NET 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、AWS SDK for .NET が認証情報プロファイル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では、Active Directory Federation Services (AD FS) を通じてフェデレーティッドユーザーアカウントを使用し、Security Assertion Markup Language (SAML) を使用することによって 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 時間有効であり、アプリケーションで AWS サービスを呼び出す際に使用されます。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 (ドメイン結合されていないアカウントでは、ユーザーは再認証の前に認証情報の入力を求められます)。

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。これらの認証情報は、AWS リソースへの制限されたアクセスの提供に加えて、指定された期間後に失効す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を介して AWS と通信する場合、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。

ユーザーとロールに関する追加情報

AWSで.NET 開発を行ったり、AWSで、NET アプリケーションを実行したりするには、これらのタスクに適したユーザー、アクセス許可セットと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、AWS IAM Identity Center でユーザーとアクセス許可セットを作成し、ID ソースから提供される一時的な認証情報を使用するの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 SDK 認証の設定 で付与されたユーザーを使用できます。適切な AWS Management Console アクセス許可があれば、そのユーザー用の最小特権のアクセス許可で別のアクセス許可セットを作成するか、開発プロジェクト専用の新しいユーザーを作成して、最小特権のアクセス許可セットを提供できます。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 Identity Centerユーザーガイド」の「はじめに」を参照してください。

サービスロール

AWS サービスロールを作成すると、ユーザーに代わって AWS サービスにアクセスできます。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. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) 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. [Next: Permissions] (次へ: アクセス権限) を選択し、アプリケーションが使用する AWS サービスに適切な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。

タグに関する情報については、IAM ユーザーガイドの「AWS リソースタグ使用したアクセスコントロール」を参照してください。

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
- AWS SDK for .NET の設定ファイルリファレンス

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「.NET Core を使用した AWS SDK for .NET の設定」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 で AWS SDK for .NET を簡単に利用するために、AWSSDK.Extensions.NETCore.Setup NuGet パッケージを使用できます。これは多くの .NET Core ライブラリと同様に `IConfiguration` インターフェイスに拡張メソッドを追加して、AWS 設定をシームレスに取得できるようにします。

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。`EnvironmentName` が Production に設定された Amazon EC2 インスタンスをデプロイする場合にはこのファイルは無視され、AWS SDK for .NET は Amazon EC2 インスタンスに設定された IAM 認証情報とリージョンにフォールバックします。

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>
```

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

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 パッケージによって、AWS サービスを依存関係インジェクションに追加するために使用できる新しい拡張メソッドが `IServiceCollection` に追加されます。次のコードは、`IConfiguration` から読み取った AWS のオプションを追加して、Amazon S3 と DynamoDB をサービスのリスト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 のサービスによって生成されたエンドポイント

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。これらは AWS SDK for .NET API でも設定できますが、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。これがデフォルトのトランスコードプリセットです。

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です。

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 を使用して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です。

AWSRegion

明示的にリージョンを指定していないクライアントのデフォルト AWS 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";
```

特定のリージョン用の AWS クライアントを作成する方法の詳細については、「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です。

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。これがデフォルトのトランスコードプリセットです。

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 を使用してサービスの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 を使用してテーブル名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 を使用して署名バージョン 4 の署名を設定するには、AWSConfigs.S3UseSignatureVersion4 プロパティを true に設定します。

```
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です。endpoint.json ファイルは、https://github.com/aws/aws-sdk-net/blob/master/sdk/src/Core/endpoints.json から入手できます。

AWS のサービスによって生成されたエンドポイント

一部の AWS サービスはリージョンの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例を 2 つ挙げると、Amazon CloudSearch と AWS IoT があります。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

AWS IoT のエンドポイントを取得するには、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");
}
```

AWS SDK for .NET の設定ファイルリファレンス

> **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 ファイルまたは Web.config ファイルを使用すると、AWS の認証情報、ログオプション、AWS サービスエンドポイント、AWS リージョンなどの AWS の設定や、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 ファイルまたは Web.config ファイルの <appSettings> 要素を引き続き使用して AWS の設定を指定することもできますが、このトピックで後ほど説明するように <configSections> 要素および <aws> 要素を使用することをお勧めします

<appSettings> エレメントの詳細については、「AWS SDK for .NET アプリケーションの設定」の <appSettings> エレメントの例を参照してください。

Note

以下の AWSConfigs クラスプロパティをコードファイルで使用して AWS の設定を指定することはできますが、以下のプロパティは非推奨であり、将来のリリースではサポートされなくなる可能性があります。

- DynamoDBContextTableNamePrefix
- EC2UseSignatureVersion4
- LoggingOptions
- LogMetrics
- ResponseLoggingOption
- S3UseSignatureVersion4

一般に、このトピックでこれから説明するように、コードファイルで AWSConfigs クラスのプロパティを使用して AWS の設定を指定するのではなく、App.config ファイルまたは Web.config ファイルの <configSections> 要素および <aws> 要素を使用して AWS の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

AWS の設定は、App.config または Web.config ファイルの <aws> 要素内で指定します。<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

AWS 設定セクションで使用できる要素のリストを次に示します。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

- map
- 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 の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の絶対パスです。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

サービスの呼び出しに使用される、保存された AWS 認証情報のプロファイル名です。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 SDK に共有される認証情報ファイルの場所の絶対パスです。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

明示的にリージョンを指定していないクライアントで使用されるデフォルトの AWS リージョン ID です。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

ログを記録する場所を示します (AWS SDK for .NET の `Amazon.AWSConfigs.LoggingConfig.LogTo` プロパティから `LogTo` 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。

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" />
```

map

`<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、AWS SDK for .NET 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) 詳細については、「DynamoDB SDK の機能強化」および「DynamoDB の属性」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 Identity Center 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、専用ソフトウェアの開発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 Identity Center 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、SDK 認証の設定 の説明に従って AWS IAM Identity Center を使用してください。

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

- お使いの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` ファイル中に格納された認証情報は平文で格納される点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全般的な説明については、「AWS 全般のリファレンス」の「AWS セキュリティ認証情報」、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

(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 SDK で使用できます。また、AWS CLI、AWS Tools for Windows PowerShell、AWS toolkits for Visual Studio、JetBrains、VS Code でも使用できます。

> **Warning**
>
> セキュリティリスクを避けるため、専用ソフトウェアの開発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 Identity Center 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、SDK 認証の設定 の説明に従って AWS IAM Identity Center を使用してください。

一般情報

共有 AWS 認証情報ファイルは、デフォルトではホームディレクトリ内の `.aws` ディレクトリに `credentials` という名前で配置されます。すなわち、`~/.aws/credentials` (Linux または macOS) または `%USERPROFILE%\.aws\credentials` (Windows) です。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、特定の形式に従います。AWS 認証情報ファイルの形式の詳細については、AWS SDK とツールの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。

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。
- このトピックの後半の記載に従い、AWS SDK for .NET API の Amazon.Runtime.CredentialManagement 名前空間を使用します。
- AWS Tools for PowerShell および AWS toolkits for Visual Studio、JetBrains、VS Code のコマンドと手順を使用します。
- 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 認証情報ファイル内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。ここは、他のプロファイルが定義されていない場合に AWS SDK for .NET が認証情報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、プログラムを使用してプロファイルを作成し、共有 AWS 認証情報ファイル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、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 リージョンを他の場所に設定したり、他の方法で設定したりすることもできます。詳細については、「AWS リージョンを設定する」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 の場合、プロファイルを作成して AWS SDK for .NET アプリケーション用に暗号化された認証情報を保存するもう 1 つの場所として SDK ストアがあります。これは `%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、専用ソフトウェアの開発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 Identity Center 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、SDK 認証の設定 の説明に従って AWS IAM Identity Center を使用してください。

一般情報

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。
- このトピックの後半の記載に従い、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 リージョンを他の場所に設定したり、他の方法で設定したりすることもできます。詳細については、「AWS リージョンを設定する」を参照してください。

AWS SDK for .NET の機能

このセクションでは、アプリケーションを作成する際に考慮する必要がある場合がある AWS SDK for .NET の機能のについて説明します。

まず、プロジェクトがセットアップされていることを確認します。

特定の AWS のサービス向けのソフトウェア開発とコード例については、「 AWS サービスの使用」を参照してください。その他のコード例については、「AWS SDK for .NET コード例」を参照してください。

トピック

- .NET の AWS 非同期 API
- 再試行とタイムアウト
- ページネーター
- その他のツール

.NET の AWS 非同期 API

AWS SDK for .NET では、比同期処理の実装のためにタスクベースの非同期パターン (TAP) 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に対する呼び出しで 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>` クラスに基づいた処理です。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に対する非同期呼び出しの完全な例については、クイックツアーをするのセクション (「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」) と「ガイダンス付きのコード例」を参照してください。

再試行とタイムアウト

AWS SDK for .NET では、AWS のサービスへの 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。

- ネットワーク接続の速度が低下している場合、または AWS のサービスに到達できない場合に、AWS SDK for .NET およびアプリケーションがどのように対応するか。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 では、サーバー側のスロットリングまたは接続中断によって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` は、Amazon.Runtime.RequestRetryMode 列挙で定義されている 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 AWS config ファイルの retry_mode 設定によって制御されます。

- `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 AWS config ファイルの max_attempts 設定によって制御されます。

これらのプロパティの詳細については、 AWS SDK for .NET API リファレンスの抽象 Amazon.Runtime.ClientConfig クラスを参照してください。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

Behavior

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の作成時に AmazonDynamoDBConfig や AmazonSQSConfig などの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で、リクエストのタイムアウトとソケットの読み取り/書き込みのタイムアウト値を設定できます。これらの値は、 `Timeout`および抽象 Amazon.Runtime.ClientConfig クラスの`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HttpWebRequestオブジェクトによって作成されたオブジェクトの プロパティ`Timeout`と `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、 HttpClient クラスではなく `HttpWebRequest` クラスを使用し、Timeout 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。

- `Timeout` 呼び出されるメソッドが AmazonS3ClientPutObjectAsyncUploadPartAsyncAmazonS3Client.() などのストリームをアップロードする場合、 と `ReadWriteTimeout`は最大値に設定されます。 AmazonGlacierClientUploadArchiveAsync
- .NET Framework をターゲットAWS SDK for .NETとする のバージョンは、 `Timeout`と をすべての AmazonS3Client とAmazonGlacierClientオブジェクトの最大値`ReadWriteTimeout`に設定します。
- 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 と AmazonGlacierClient オブジェクトの最大値`Timeout`に設定されます。

例

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
  });
```

ページネーター

一部の AWS サービスでは大量のデータを収集して格納します。このデータは、AWS SDK for .NET の API コールを使用して取得できます。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。

例えば、 AmazonCloudWatchLogsClient クラスの定義を調べると、 Paginatorsプロパティが表示されます。これは、Amazon CloudWatch Logs 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。。

例えば、AmazonCloudWatchLogsClient前述の では、 Paginator オブジェクトには、 API コールの完全な DescribeLogGroupsResponse オブジェクトを含む Responsesプロパティが含まれています。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。.NET Framework 4.5 (またはそれ以降) のプロジェクトでは、同期ページ分割を使用できます。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。同期ページネーションと.NET Framework 4.5 (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、例の後の ページネーターに関する追加の考慮事項 で説明します。

例

次の例では、AWS SDK for .NET を使用してロググループのリストを表示する方法を示します。比較のために、この例では、ペー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しない CloudWatch ロググループの取得

```
// 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た CloudWatch ロググループの取得

```
      // 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は `IAsyncEnumerable`インターフェイスを使用するため、Microsoft.Bcl.AsyncInterfaces NuGet package が必要に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 AmazonCloudWatchLogs。モデル

  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で特定の AWS ページネーターの結果が必要な場合でも、ページネーターオブジェクトを複数回使用することはできません。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

  .NET Framework 4.5 (またはそれ以降) のプロジェクトでは、同期ページ分割を使用できます。

  > **Warning**
  >
  > 2024 年 8 月 15 日以降、 AWS SDK for .NETは .NET Framework 3.5 のサポートを終了し、.NET Framework の最小バージョンを 4.6.2 に変更します。詳細については、ブログ記事「 の .NET Framework 3.5 および 4.5 ターゲットに関する重要な変更AWS SDK for .NET点」を参照してください。

  これを確認するには、.NET Framework 4.5 (またはそれ以降) のプロジェクトを作成して、上記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。

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.NET Core アプリケーションを開発したら、.NET CLI 用の AWS デプロイツールを使用して、より簡単にアプリケーションを AWS にデプロイできます。

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

AWS Message Processing Framework for .NET

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。

Amazon SQS、Amazon SNS、Amazon などのサービスを使用している場合は EventBridge、.NET 用の AWS Message Processing Framework を活用できる場合があります。詳細については、「AWS .NET 用の Message Processing Framework」を参照してください。

AWS SDK for .NETによる高度な認証と承認

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承認の高度な手法について説明します。

トピック

- AWS SDK for .NETでのシングルサインオン (SSO)

AWS SDK for .NETでのシングルサインオン (SSO)

AWS IAM Identity Center は、AWS アカウント とクラウドアプリケーションへの SSO アクセスの一元管理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. AWSconfigユーザーのコンピュータ上の共有ファイルが SSO 情報で更新されます。
3. ユーザーは IAM Identity Center からサインインすると、設定されている AWS Identity and Access Management (IAM) 権限の短期認証情報が与えられます。このサインインは、AWS CLI のような非 SDK ツールを使って開始することも、.NET アプリケーションを通じてプログラム的に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 Identity Centerの使用に関する参考情報を提供します。SDK 認証の設定での基本的な 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。AWSでSSO を初めて使用する場合は、まずそのトピックで基本的な情報を確認し、次に次のチュートリアルで SSO の動作を確認するとよいでしょう。

- 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

IAM Identity Center を使用する前に、ID ソースの選択や関連するAWS アカウントや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AWS アカウントで、IAM Identity Center を設定したら、SSO 用の名前付きプロファイルをユーザーの共有AWSconfig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が含まれる AWS リージョン。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 の完全なリストとそれらのコードについては、Amazon Web Services 全般のリファレンス の 「リージョナルエンドポイント」を参照してください。

sso_account_id

AWS Organizations サービスを通じて追加された AWS アカウントの 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、次にそのトークンを使用して適切なAWSアプリケーションや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。
- AWS CLIを使用してトークンを生成し、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。

AWS CLIを使用するほかに、SSO トークンをプログラムでも生成できます。

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。

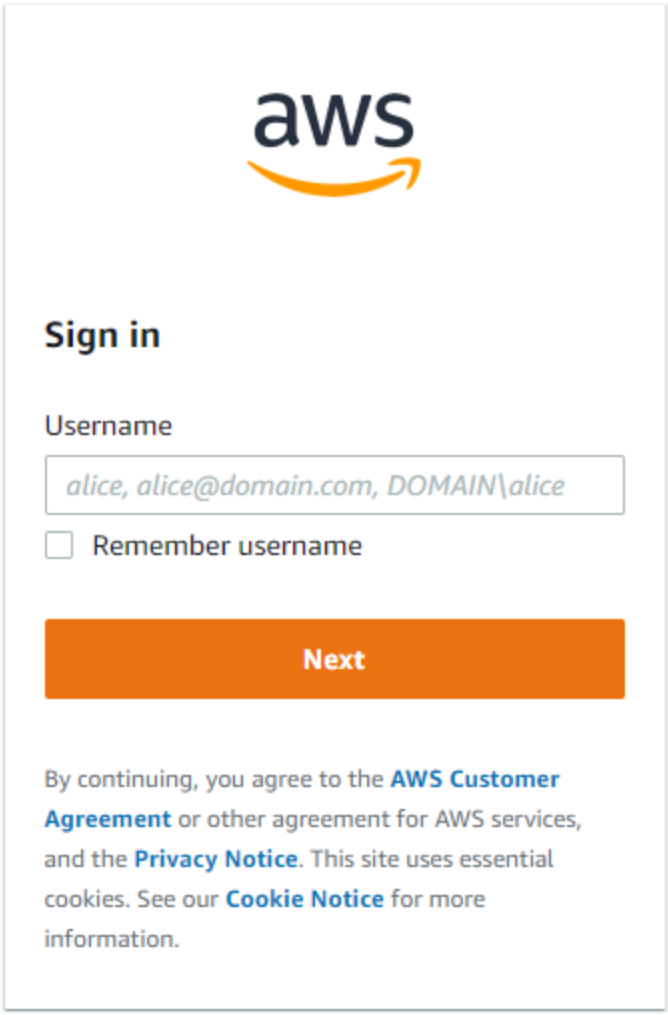
次のコードスニペットは、このメソッドを示しています。

> **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
```

```
        });
    };

    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、AWS CLIを使用して一時的な SSO トークンを生成する方法と、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する SSO のチュートリアルを参照してください。

AWS CLIを使用して SSO トークンを生成します。

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。その場合、アプリケーションは、「var client = new AmazonS3Client();」と同様に、パラメーターなしで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する 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 を使用するためのAWS CLIの設定
- AWS Toolkit for Visual Studioで IAM Identity Centerの認証情報を使用する

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する 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、AWS SDK for .NETの SSO 機能の一部を紹介します。IAM Identity CenterをAWS SDK for .NETと併用する方法の詳細については、背景情報を含むトピック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略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手順は、AWS Organizations や IAM Identity Center などのサービスの設定に役立ちます。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、ツールチェーンのインストールと設定や開始方法などのセクションで説明しています。
- SSO のテストに使用できるものを少なくとも 1 つのAWS アカウントを特定または作成してください。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 ユーザーを特定してください。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 をセットアップする

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を説明します。

この設定を行うには、まず管理者としてテストAWS アカウン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、Organizations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。

このアクションにより、テストAWS アカウントが管理アカウントとして組織に追加されます。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 ID 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にする」を参照してください。

次に、以下の設定を実行します。

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、`sso_start_url`その値を後で使用できるように設定に記録します。
2. のバナーでAWS Management Console、SSO AWS リージョン を有効にしたときに設定されたものを探してください。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. 以下のように 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。次に、[次へ] 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許可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。

```
{
    "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、SSOReadOnlyRole権限セット名としてを入力します。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。設定には数秒かかります。

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。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および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
                });
            };

            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および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
                });
            };

            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ーが存在しない場合は、作成してください。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ーに共有AWS configファイルを作成し、以下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. 表示されるウェブ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、AWS のリソースまたは開発環境内のリソース (ファイルやフォルダなど) である可能性があります。

AWS CLIおよび.NET アプリケーションを使用する 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 を有効にする方法を示します。プログラム的に生成するのでなく、AWS CLIを使用して一時的な SSO トークンを生成します。

このチュートリアルでは、AWS SDK for .NETの SSO 機能の一部を紹介します。IAM Identity CenterをAWS SDK for .NETと併用する方法の詳細については、背景情報を含むトピック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大まかな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 などのサービスの設定に役立ちます。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、ツールチェーンのインストールと設定や開始方法などのセクションで説明しています。
- SSO のテストに使用できる少なくとも 1 つのAWS アカウントを指定または作成してください。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 ユーザーを指定してください。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 フレームワークがインストールされている必要があります。

- SSO ユーザーのコンピューターに、AWS CLIバージョン 2 がインストールされていることを確認してください。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。

AWS をセットアップする

このセクションでは、このチュートリアルでさまざまなAWSサービスを設定する方法を説明します。

この設定を行うには、まず管理者としてテストAWS アカウン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、Organizations 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。

このアクションにより、テストAWS アカウントが管理アカウントとして組織に追加されます。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 ID 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 アイデンティティーセンター ユーザーガイド」の「IAM アイデンティティー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 Identity Center を設定

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. AWS Management Consoleのバナーで、SSO を有効にしたときに設定されたAWS リージョンを探してください。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、sso_region設定にリージョンコードを記録しておきます。このコードはus-east-1 のようになります。
3. SSO ユーザーを次のように作成します。
   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。次に、[次へ] 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. [権限セットタイプ] 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。

```
{
    "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、sso_role_name設定に権限セットの名前を設定に記録します。

6. AWSアカウントページに移動し、すでに組織に追加してある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

次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 メールを確認して SSO 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AWSconfigファイルを含むフォルダーが存在しない場合は、作成してください。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ーに共有AWSconfig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. 以下の 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たくない場合は、リソースをクリーンアップします。これらは、AWS のリソースまたは開発環境内のリソース (ファイルやフォルダなど) である可能性があります。

アプリケーションのAWSへのデプロイ

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。これは、AWS Management Console または AWS CloudFormation や AWS Cloud Development Kit (AWS CDK) などの特定のサービスを使用して行うことができます。また、デプロイメント用に作成された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 用の次のAWSツールを使用して、アプリケーションをAWSにデプロイできます。

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

AWSツールキットを使用して、任意の 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「AWS に公開」機能は、.NET CLI のデプロイ 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 StudioDD 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操作」および「AWS App Runner の操作」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 RunnerBB 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 を使用してこれらの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。

- アプリケーションに適したコンピューティングの推奨事項-コンピューティングに関する推奨事項を確認して、どのAWSコンピューティングがアプリケーションに最も適しているかを調べてください。
- 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 — 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ーをプロビジョニングし、選択したAWSコンピュートにアプリケーションをデプロイします。
- 繰り返し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。また、プロジェクトをバージョン管理してチームと共有し、繰り返しデプロイできるようにすることもできます。
- BB for .NET のAWS CDK学習を支援 - このツールは、AWS CDK など、その構築の基礎となる AWS ツールを徐々に学習するのに役立ちます。

AWS デプロイ ツールは、次の AWS サービスへの ASP.NET Core アプリケーションのデプロイをサポートしています。

- AWS Fargate を使用した Amazon ECS サービス - 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 Web 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、[ドキュメント]、[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 AWS用のデプロイツールを使用すると、.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 にデプロイでき、デプロイプロセスをガイドしてくれます。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。

- アプリケーションに適したコンピューティングの推奨事項-コンピューティングに関する推奨事項を確認して、どのAWSコンピューティングがアプリケーションに最も適しているかを調べてください。
- 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 — 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ーをプロビジョニングし、選択したAWSコンピュートにアプリケーションをデプロイします。
- 繰り返し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。また、プロジェクトをバージョン管理してチームと共有し、繰り返しデプロイできるようにすることもできます。
- .NET用 AWS CDK の学習に役立つ-このツールを使用すると、AWS構築の基盤となるツール (など) を徐々に学習できます。AWS CDK

AWSDeploy Tool は、.NET Console アプリケーションを以下のAWSサービスにデプロイすることをサポートします。

- AWS Fargateを使用した Amazon ECS サービス - 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) へのサービス (バックグラウンド プロセッサなど) としての .NET 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 スケジュールタスク 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 プロセスなど) として .NET アプリケーションを Amazon ECS にデプロイできます。

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、[ドキュメント]、[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デプロイツールは、コンテンツネットワーク配信 CloudFront に Amazon を使用して、Blazor WebAssembly アプリケーションを Amazon S3 でホストするのに役立ちます。Amazon S3 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 — 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ーをプロビジョニングし、選択したAWSコンピュートにアプリケーションをデプロイします。
- 繰り返し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。また、プロジェクトをバージョン管理してチームと共有し、繰り返しデプロイできるようにすることもできます。
- .NET用AWS CDK の学習を支援- このツールは、AWS CDK など、その構築の基礎となる AWS ツールを徐々に学習するのに役立ちます。

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、[ドキュメント]、[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、サーバーをプロビジョニングまたは管理せ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

.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 Core CLI」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は、`--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

以下の手順は、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、プロジェクトには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 関数のデプロイ先になるリージョン) 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 にすでに存在する関数の名前、またはまだ 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。

AWS SDK for .NET 用にプロジェクトを移行する

このセクションでは、ユーザーに適用される可能性のある移行タスクと、それらのタスクの実行方法について説明します。

トピック

の最新情報 AWS SDK for .NET

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

AWS SDK for .NETの最新情報は次のとおりです。

2024 年 3 月 28 日: AWS Message Processing Framework for .NET のプレリリース

> 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。

AWS Message Processing Framework for .NET は、Amazon Simple Queue Service (SQS)、Amazon Simple Notification Service (SNS)、Amazon 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 EventBridge。

2024 年 2 月 23 日: .NET 8 のサポートを追加

.NET 8 のサポートが に追加されました AWS SDK for .NET。.NET 8 以降の をサポートする最新のNuGet パッケージまたはアセンブリを使用してください。 ???Lambda のサポートなど、このサポートに関する追加情報については、 のブログ記事「.NET 8 サポート」を参照してください AWS。

2024 年 2 月 18 日: .NET Framework サポートへの今後の変更

2024 年 8 月 15 日以降、 AWS SDK for .NET は .NET Framework 3.5 のサポートを終了し、.NET Framework の最小バージョンを 4.6.2 に変更します。詳細については、ブログ記事「 の .NET Framework 3.5 および 4.5 ターゲットに関する重要な変更 AWS SDK for .NET点」を参照してください。

2023-07-17: AWS Lambda Annotations フレームワークが一般提供のためにリリースされました

AWS Lambda アノテーションフレームワークでは、C# ソースジェネレーターテクノロジーを使用することで、.NET 開発者が C# で Lambda 関数を記述するのがより自然に感じられるようになります。一般提供が開始されました。

2023-07-15: DynamoDB の分散キャッシュプロバイダーがプレビュー版でリリースされました

> 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。

分散キャッシュプロバイダーライブラリにより、Amazon DynamoDB を ASP.NET Core の分散キャッシュフレームワークのストレージとして使用できます。詳細については、ブログ記事「Introducing the AWS .NET Distributed Cache Provider for DynamoDB (Preview) and the GitHub repository」を参照してください。

2022-07-13: AWS デプロイツールがリリースされました

AWS デプロイツールがリリースされました。このツールは、.NET CLI と のインタラクティブなツール AWS Toolkit for Visual Studio であり、最小限の AWS 知識で、クリックやコマンドを最小限に抑えて .NET アプリケーションをデプロイするのに役立ちます。詳細については、「アプリケーションのAWSへのデプロイ」を参照してください。

2020-08-24: バージョン 3.5 の SDK がリリースされました

- SDK のすべての非 Framework バリエーションに対するサポートを .NET Standard 2.0 に移行することで、.NET エクスペリエンスの標準化を進めました。詳細については、「バージョン 3.5 への移行」を参照してください。
- 多くのサービスクライアントにページネーターが追加され、API の結果のページ分割がより便利になりました。詳細については、「ページネーター」を参照してください。

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 には、さまざまなプラットフォームを対象とする開発者向けに、異なるアセンブリのグループが用意されています。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

AWS SDK for .NET は、.NET Core (.NET Core 3.1、.NET 5、.NET 6 など) 用に作成されたアプリケーションをサポートします。 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

AWS SDK for .NET の非 Framework バリエーションは .NET Standard 2.0 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

> **Warning**
>
> 2024 年 8 月 15 日以降、 AWS SDK for .NETは .NET Framework 3.5 のサポートを終了し、.NET Framework の最小バージョンを 4.6.2 に変更します。詳細については、ブログ記事「 の .NET Framework 3.5 および 4.5 ターゲットに関する重要な変更点AWS SDK for .NET」を参照してください。

このバージョンの AWS SDK for .NET は、.NET Framework 4.5 でコンパイルされ、.NET 4.0 ランタイムで実行されます。AWS サービスクライアントは、同期または非同期の呼び出しパターンをサポートし、C# 5.0 で導入された async および await キーワードを使用します。

.NET Framework 3.5

> **Warning**
>
> 2024 年 8 月 15 日以降、 AWS SDK for .NETは .NET Framework 3.5 のサポートを終了し、.NET Framework の最小バージョンを 4.6.2 に変更します。詳細については、ブログ記事「 の .NET Framework 3.5 および 4.5 ターゲットに関する重要な変更点AWS SDK for .NET」を参照してください。

このバージョンの AWS SDK for .NET は、.NET Framework 3.5 でコンパイルされ、.NET 2.0 または .NET 4.0 ランタイムで実行されます。AWS サービスクライアントは、同期または非同期の呼び出しパターンをサポートし、従来の Begin および End パターンを使用します。

> **Note**
>
> AWS SDK for .NET は、CLR のバージョン 2.0 でビルドされたアプリケーションで使用する場合には、連邦情報処理規格 (FIPS) に準拠していません。その環境で FIPS 準拠の実装を置き換える方法の詳細については、Microsoft ブログCryptoConfigの「」と Security.Cryptography.dll の CLR Security チームの HMACSHA256 クラス (HMACSHA256Cng) を参照してください。

ポータブルクラスライブラリと Xamarin

AWS SDK for .NET には、ポータブルクラスライブラリの実装も含まれています。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、非同期の呼び出しパターンのみをサポートしています。

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。

詳細情報

バージョン 3.5 の AWS SDK for .NET への移行

バージョン 3 の AWS SDK for .NET への移行

このトピックでは、AWS SDK for .NET のバージョン 3 での変更点、およびこのバージョンの SDK へのコードの移行方法について説明します。

AWS SDK for .NET のバージョンについて

当初、AWS SDK for .NET は 2009 年 11 月にリリースされ、.NET Framework 2.0 向けに設計されていました。このリリース以降、.NET は .NET Framework 4.0 および .NET Framework 4.5 で改善され、新しい対象プラットフォームとして WinRT と Windows Phone が追加されています。

AWS SDK for .NET バージョン 2 では、.NET プラットフォームの新機能を利用するように更新され、WinRT と Windows Phone も対象プラットフォームとして追加されました。

AWS SDK for .NET バージョン 3 ではアセンブリがモジュール化されました。

SDK のアーキテクチャの再設計

バージョン 3 の AWS SDK for .NET 全体がモジュラー式に再設計されています。1 つの大きなアセンブリとしてではなく、各サービスが個別のアセンブリとして実装されます。AWS SDK for .NET 全体をアプリケーションに追加する必要はなくなりました。アプリケーションで使用する AWS サービスのアセンブリだけを追加できます。

破壊的変更

以下のセクションでは、バージョン 3 の AWS SDK for .NET への変更点について説明します。

AWSClientFactory の削除

`Amazon.AWSClientFactory` クラスは削除されました。現在、サービスクライアントを作成するには、サービスクライアントのコンストラクタを使用します。たとえば、`AmazonEC2Client` を作成するには:

```
var ec2Client = new Amazon.EC2.AmazonEC2Client();
```

Amazon.Runtime.AssumeRoleAWSCredentials の削除

`Amazon.Runtime.AssumeRoleAWSCredentials` クラスは、コア名前空間内にありながら AWS Security Token Service に依存していたこと、および長い間 SDK で使用されていなかったことか

ら、削除されました。代わりに、Amazon.SecurityToken.AssumeRoleAWSCredentials クラスを使用してください。

S3Link からの SetACL メソッドの削除

S3Link クラスは Amazon.DynamoDBv2 パッケージの一部であり、DynamoDB 項目内の参照であるオブジェクトを Amazon S3 に保存するために使用されています。これは便利な機能ですが、Amazon.S3 パッケージに DynamoDB に対するコンパイル依存関係を作成するのは好ましくありませんでした。結果として、S3Link クラスで公開されている Amazon.S3 メソッドを簡素化し、SetACL メソッドを MakeS3ObjectPublic メソッドに置き換えました。オブジェクトでアクセスコントロールリスト (ACL) を細かく制御する場合は、Amazon.S3 パッケージを使用します。

サポートされなくなった結果クラスの削除

AWS SDK for .NET のほとんどのサービスでは、操作は、リクエスト ID や結果オブジェクトなどの操作のメタデータを含む応答オブジェクトを返します。応答クラスと結果クラスを分けておくと、冗長であり、開発者は余分な入力が必要でした。AWS SDK for .NET のバージョン 2 では、結果クラスのすべての情報をレスポンスクラス内に移しました。また、結果クラスをサポート対象外として、その使用を非推奨にしました。AWS SDK for .NET のバージョン 3 では、これらのサポートされなくなった結果クラスを削除して SDK のサイズを減らしました。

AWS Config セクションの変更

App.config または Web.config ファイルを使用して、AWS SDK for .NET の詳細設定を行うことができます。これは、SDK アセンブリ名を参照する、次のような <aws> config セクションを通じて行います。

```
<configuration>
  <configSections>
    <section name="aws" type="Amazon.AWSSection, AWSSDK"/>
  </configSections>
  <aws region="us-west-2">
    <logging logTo="Log4Net"/>
  </aws>
</configuration>
```

AWS SDK for .NET のバージョン 3 では、AWSSDK アセンブリは存在しなくなりました。共通コードは AWSSDK.Core に格納しました。そのため、App.config ファイルまたは Web.config ファイルでの AWSSDK アセンブリへの参照を、次のように AWSSDK.Core アセンブリを参照するように変更する必要があります。

```
<configuration>
  <configSections>
    <section name="aws" type="Amazon.AWSSection, AWSSDK.Core"/>
  </configSections>
  <aws region="us-west-2">
    <logging logTo="Log4Net"/>
  </aws>
</configuration>
```

また、Amazon.AWSConfigs クラスで構成設定を操作できます。AWS SDK for .NET のバージョン 3 では、DynamoDB の構成設定を Amazon.AWSConfigs クラスから Amazon.AWSConfigsDynamoDB クラスに移しました。

バージョン 3.5 の AWS SDK for .NET への移行

バージョン 3.5 の AWS SDK for .NET は、SDK のすべての非 Framework バリエーションに対するサポートを .NET Standard 2.0 に移行することで、.NET エクスペリエンスの標準化を進めます。環境とコードベースによっては、バージョン 3.5 の機能を利用するために、特定の移行作業が必要になる場合があります。

このトピックでは、バージョン 3.5 の変更点と、環境やコードをバージョン 3 から移行するために必要な作業について説明します。

バージョン 3.5 の変更点

AWS SDK for .NET バージョン 3.5 で変更された点と変更されていない点について以下に説明します。

.NET Framework と .NET Core

.NET Framework と .NET Core に対するサポートは変更されていません。

Xamarin

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。

Unity

Unity アプリは、Unity 2018.1 以降を使用する .NET Standard 2.0 または .NET 4.x プロファイルを対象とする必要があります。詳細については、「.NET profile suppot」を参照してください。さらに、IL2CPP を使用してビルドする場合は、link.xml ファイルを追加してコードストリッピングを無効にする必要があります。詳細については、「Referencing the AWS SDK for .NET Standard 2.0 from Unity, Xamarin, or UWP」を参照してください。推奨されているコードベースのいずれかにコードを移植すると、SDK が提供するすべてのサービスに Unity アプリからアクセスできます。

Unity は .NET Standard 2.0 をサポートするため、SDK バージョン 3.5 の AWSSDK.Core パッケージから Unity 固有のコードが除外されました。一部の上位レベルの機能も除外されました。すべての レガシー Unity コードは aws/aws-sdk-unity-net GitHub リポジトリで参照可能であり、移行に役立てることができます。Unity での AWS の使用に関連する機能が不足している場合は、https://github.com/aws/dotnet/issues で機能のリクエストを申請できます。

また、「Unity のサポートに関する特別な考慮事項」も参照してください。

ユニバーサル Windows プラットフォーム (UWP)

UWP アプリケーションの対象はバージョン 16299 以降 (2017 年 10 月リリースの Fall Creators Update、バージョン 1709) とします。

Windows Phone と Silverlight

これらのプラットフォームは、Microsoft で現在開発されていないため、バージョン 3.5 の AWS SDK for .NET ではサポートされません。詳細については、次を参照してください。

- Windows 10 Mobile のサポート終了
- Silverlight のサポート終了

レガシーポータブルクラスライブラリ (プロファイルベースの PCL)

ライブラリを .NET Standard に再ターゲットすることを検討します。詳細については、「ポータブルクラスライブラリとの比較」を参照してください。

Amazon Cognito Sync マネージャーと Amazon Mobile Analytics マネージャー

Amazon Cognito Sync と Amazon Mobile Analytics の使用を容易にする高レベルの抽象化は、バージョン 3.5 の AWS SDK for .NET から削除されています。Amazon Cognito Sync の代わりに AWS

AppSync の使用が推奨されます。Amazon Mobile Analytics の 代わりに Amazon Pinpoint の使用が推奨されています。

AWS AppSync および Amazon Pinpoint の上位レベルのライブラリコードの欠如によってコードが影響を受ける場合は、GitHub 問題 (https://github.com/aws/dotnet/issues/20 と https://github.com/aws/dotnet/issues/19) のいずれかまたは両方で報告できます。また、Amazon Cognito Sync マネージャーと Amazon Mobile Analytics マネージャーのライブラリは、GitHub リポジトリ (aws/amazon-cognito-sync-manager-net と aws/aws-mobile-analytics-manager-net) から入手できます。

同期コードの移行

AWS SDK for .NETのバージョン 3.5 は、.NET フレームワークと.Net Standard (.NET Core 3.1、.NET 5 などの.NET Core バージョンによる) の両方をサポートしています。.Net Standardに準拠する SDK のバリエーションでは、非同期メソッドしか提供されないため、.NET Standard を利用するには、同期コードを変更して非同期で実行する必要があります。

次のコードスニペットは、同期コードを非同期コードに変更する方法を示しています。これらのスニペットのコードでは、Amazon S3 バケットの数を表示します。

元のコードは ListBuckets を呼び出します。

```
private static ListBucketsResponse MyListBuckets()
{
  var s3Client = new AmazonS3Client();
  var response = s3Client.ListBuckets();
  return response;
}

// From the calling function
ListBucketsResponse response = MyListBuckets();
Console.WriteLine($"Number of buckets: {response.Buckets.Count}");
```

バージョン 3.5 の SDK を使用するには、代わりに ListBucketsAsync を呼び出します。

```
private static async Task<ListBucketsResponse> MyListBuckets()
{
  var s3Client = new AmazonS3Client();
  var response = await s3Client.ListBucketsAsync();
  return response;
}
```

```
// From an **asynchronous** calling function
ListBucketsResponse response = await MyListBuckets();
Console.WriteLine($"Number of buckets: {response.Buckets.Count}");

// OR From a **synchronous** calling function
Task<ListBucketsResponse> response = MyListBuckets();
Console.WriteLine($"Number of buckets: {response.Result.Buckets.Count}");
```

バージョン 3.7 の AWS SDK for .NET への移行

バージョン 3.7 より、AWS SDK for .NET は .NET Standard 1.3 をサポートしなくなりました。

.NET Standard 1.3 からの移行については、「.NET Standard 1.3 からの移行」を参照してください。

.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。この発表に従い、AWS は 2020 年 12 月 31 日に、AWS SDK for .NET での .NET Standard 1.3 のサポートを終了しました。

AWS では 2020 年 10 月 1 日まで、.NET Standard 1.3 を対象とした AWS SDK for .NET のサービス更新プログラムとセキュリティ修正プログラムの提供を継続してきました。この日以降、.NET Standard 1.3 を対象とする製品はメンテナンスモードに入り、新しい更新プログラムはリリースされなくなりました。AWS は重大なバグ修正とセキュリティパッチの適用のみを行いました。

2020 年 12 月 31 日に、AWS SDK for .NET での .NET Standard 1.3 のサポートが終了しました。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。

での AWS サービスの操作 AWS SDK for .NET

以下のセクションには、 を使用して AWS サービスを操作する方法を示す例、チュートリアル、タスク AWS SDK for .NET 、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、まずクイックツアーをするトピックをチェックすることをお勧めします。SDK についてわかりやすく説明しています。

その他のコード例は、 のAWS 「コード例リポジトリ」と「awslabs」リポジトリにありますGitHub。

開始する前に、環境とプロジェクトがセットアップされている ことを必ず確認してください。また、「SDK の機能」の情報を確認してください。

トピック

- のガイダンスを含むコード例 AWS SDK for .NET
- コンピュートサービスでAWS Lambdaを使用する
- AWS SDK for .NET の高レベルライブラリとフレームワーク
- スタックとアプリケーションを使用するための AWS OpsWorks のプログラミング
- 他の AWS サービスと設定のサポート

のガイダンスを含むコード例 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。を使用して AWS のサービス AWS SDK for .NET を操作する方法を学ぶのに役立ちます。

を初めて使用する場合は AWS SDK for .NET、まずクイックツアーをするトピックをチェックすることをお勧めします。SDK についてわかりやすく説明しています。

開始する前に、環境とプロジェクトがセットアップされている ことを必ず確認してください。また、「SDK の機能」の情報を確認してください。

トピック

- AWS CloudFormation を使用したアクセス 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

AWS CloudFormation を使用したアクセス 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 AWS CloudFormation 、クライアント用の APIs を提供します。APIs を使用すると、テンプレートやスタックなどの AWS 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。APIsAWS SDK for .NET リファレンス」を参照してください (「Amazon」までスクロール CloudFormationします)。

AWS CloudFormation APIsは . AWSSDKCloudFormation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、環境とプロジェクトがセットアップされていることを必ず確認してください。また、「SDK の機能」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、 を使用して AWS CloudFormation スタック内のリソースを AWS SDK for .NET 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 AmazonCloudFormation。モデル

  クラス I CloudFormationPaginatorFactory。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 と AWS SDK for .NET バージョン 3.3 以前に基づくプロジェクトに固有のものです。

Amazon Cognito Identity を使用すると、ユーザー用に一意の 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ユーザーを認証できます。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 この例では、 CognitoAuthentication 拡張ライブラリを使用して Amazon Cognito ユーザープールを認証する方法を示します。

トピック

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

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 と AWS SDK for .NET バージョン 3.3 以前に基づくプロジェクトに固有のものです。

`Amazon.CognitoIdentity.CognitoAWSCredentials`AWSSDKCognitoIdentity 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。

Cognito をセットアップするAWSCredentials

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 経由で認証し、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 拡張機能ライブラリの例

> **Note**
>
> このトピックの情報は、.NET Framework と AWS SDK for .NET バージョン 3.3 以前に基づくプロジェクトに固有のものです。

Amazon.Extensions.CognitoAuthentication NuGet package にある CognitoAuthentication 拡張機能ライブラリは、.NET Core および Xamarin デベロッパー向けの Amazon Cognito ユーザープールの認証プロセスを簡素化します。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。

拡張機能ライブラリの使用 CognitoAuthentication

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

署名付きリクエストを必要としない匿名 AWSCredentialsAmazonCognitoIdentityProviderClientを使用して 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Multi-Factor Authentication (MFA)フローの継続も簡単です。唯一の要件は、 CognitoAuthentication オブジェクト、SRP のユーザーのパスワード、次のチャレンジに必要な情報です。この情報は、ユーザーに入力を求めた後に取得されます。次のコードは、チャレンジタイプをチェックし、認証フロー中に MFA と NewPasswordRequired チャレンジの適切なレスポンスを取得する方法の 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、ユーザーが適切な AWS リソースにアクセスすること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、および 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

CognitoAuthentication 拡張ライブラリは、SRP NewPasswordRequired、および MFA に加えて、以下の認証フローを容易にします。

- Custom - `StartWithCustomAuthAsync(InitiateCustomAuthRequest customRequest)` を呼び出すことで開始する
- RefreshToken - への呼び出しを開始します。`StartWithRefreshTokenAuthAsync(InitiateRefreshTokenAuthRequest refreshTokenRequest)`
- RefreshTokenSRP - への呼び出しで開始する `StartWithRefreshTokenAuthAsync(InitiateRefreshTokenAuthRequest refreshTokenRequest)`
- AdminNoSRP - への呼び出しで開始する `StartWithAdminNoSrpAuthAsync(InitiateAdminNoSrpAuthRequest adminAuthRequest)`

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。

トピック

- オブジェクト永続性モデル
- 詳細情報
- 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

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。このモデルにアクセスするには、Amazon.DynamoDBv2.DocumentModel 名前空間を使用します。

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。たとえば、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。

TableBuilder

. AWSSDKDynamoDBv2 NuGet 2package のバージョン 3.7.203 では、代替メカニズムである TableBuilder クラスが導入されました。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。取得する返信を決定するために、 GetItemAsyncメソッドはターゲット返信の hash-and-range プライマリキーを使用します。

```
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。このモデルにアクセスするには、Amazon.DynamoDBv2.DataModel 名前空間を使用します。

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。たとえば、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、 属性DynamoDBHashKeyと DynamoDBRangeKey 属性はテーブルの hash-and-range 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 プロパティを含むコンテキスト

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。このプロパティは、 AWSSDK.DynamoDBv2 NuGet package のバージョン 3.7.203 で導入され、コールドスタートおよびスレッドプールの動作が原因でレイテンシーやデッドロックが増加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、.DynamoDBv2package のバージョン 3.7 AWSSDK.203 DynamoDBv2 NuGet 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

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

- 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と例

- 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操作

ドキュメントモデル情報と例

- 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と例

- .NET: オブジェクト永続性モデル

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 と AWS SDK for .NET バージョン 3.3 以前に基づく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、「ケーススタディ: ProductCatalog 項目」で提供されている例に基づいています。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、各属性値には、属性を出力する正しい形式を決定するために評価できる data-type-specific プロパティがいくつかあります。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、、 などのプロパティの場合SS、 Countが 0 M NSより大きい場合、 属性は対応する non-zero-value データ型になります。属性のすべての data-type-specific プロパティが nullまたは がゼロにCount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 と AWS SDK for .NET バージョン 3.3 以前に基づくプロジェクトに固有のものです。

は、Amazon DynamoDB を操作するときに 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。この例では、取得する項目を決定するために、 GetItemメソッドはターゲット項目の hash-and-range プライマリキーを使用します。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方法の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、Amazon EC2 クライアント用の APIs を提供します。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 APIs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、環境とプロジェクトがセットアップされていることを必ず確認してください。また、「SDK の機能」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」およびAmazon EC2 ユーザーガイド」を参照してください。

を使用する場合 AWS SDK for .NET、VPC または EC2-Classic の EC2 で使用するセキュリティグループを作成できます。VPC 内の EC2 と EC2-Classic の詳細については、Amazon EC2 ユーザーガイド」またはAmazon EC2 ユーザーガイド」を参照してください。

> **Warning**
>
> 2022 年 8 月 15 日に、EC2-Classic の提供を終了し h ます。EC2-Classic は、VPC への移行をお勧めします。詳細については、Amazon EC2 EC2-Classic から VPC への移行Amazon EC2」を参照してください。ブログ記事「EC2-Classic Networking は販売終了になります — 準備方法はこちら」も参照してください。

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
- 追加の考慮事項

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();
      }
    }
  }
}
```

追加の考慮事項

- 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プロパティの説明から取得されます。

- セキュリティグループの完全なリストを取得するには、パラメータ DescribeSecurityGroupsAsync なしで を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、 を使用してセキュリティグループ AWS SDK for .NET 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。詳細については、親セクションにある VPC 内の EC2 と EC2-Classic の違いに関する参照情報 (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。

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の作成
- 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の作成

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
{
  // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
 = = =
  // Class to create a security group
  class Program
  {
    private const int MaxArgs = 2;

    static async Task Main(string[] args)
    {
      // Parse the command line and show help if necessary
      var parsedArgs = CommandLine.Parse(args);
      if(parsedArgs.Count == 0)
      {
        PrintHelp();
        return;
      }
      if(parsedArgs.Count > MaxArgs)
        CommandLine.ErrorExit("\nThe number of command-line arguments is incorrect." +
          "\nRun the command with no arguments to see help.");

      // Get the application arguments from the parsed list
      var groupName = CommandLine.GetArgument(parsedArgs, null, "-g", "--group-name");
      var vpcID = CommandLine.GetArgument(parsedArgs, null, "-v", "--vpc-id");
      if(string.IsNullOrEmpty(groupName))
        CommandLine.ErrorExit("\nYou must supply a name for the new group." +
          "\nRun the command with no arguments to see help.");
      if(!string.IsNullOrEmpty(vpcID) && !vpcID.StartsWith("vpc-"))
        CommandLine.ErrorExit($"\nNot a valid VPC ID: {vpcID}");

      // groupName has a value and vpcID either has a value or is null (which is fine)
      // Create the new security group and display information about it
      var securityGroups =
        await CreateSecurityGroup(new AmazonEC2Client(), groupName, vpcID);
      Console.WriteLine("Information about the security group(s):");
```

```
      foreach(var group in securityGroups)
      {
        Console.WriteLine($"\nGroupName: {group.GroupName}");
        Console.WriteLine($"GroupId: {group.GroupId}");
        Console.WriteLine($"Description: {group.Description}");
        Console.WriteLine($"VpcId (if any): {group.VpcId}");
      }
    }


    //
    // Method to create a new security group (either EC2-Classic or EC2-VPC)
    // If vpcID is empty, the security group will be for EC2-Classic
    private static async Task<List<SecurityGroup>> CreateSecurityGroup(
      IAmazonEC2 ec2Client, string groupName, string vpcID)
    {
      // See if one or more security groups with that name
      // already exist in the given VPC. If so, return the list of them.
      var securityGroups = await FindSecurityGroups(ec2Client, groupName, vpcID);
      if (securityGroups.Count > 0)
      {
        Console.WriteLine(
          $"\nOne or more security groups with name {groupName} already exist.\n");
        return securityGroups;
      }

      // If the security group doesn't already exists, create it.
      var createRequest = new CreateSecurityGroupRequest{
        GroupName = groupName
      };
      if(string.IsNullOrEmpty(vpcID))
      {
        createRequest.Description = "Security group for .NET code example (no VPC
specified)";
      }
      else
      {
        createRequest.VpcId = vpcID;
        createRequest.Description = "Security group for .NET code example (VPC: " +
vpcID + ")";
      }
      CreateSecurityGroupResponse createResponse =
        await ec2Client.CreateSecurityGroupAsync(createRequest);
```

```
      // Return the new security group
      DescribeSecurityGroupsResponse describeResponse =
        await ec2Client.DescribeSecurityGroupsAsync(new DescribeSecurityGroupsRequest{
          GroupIds = new List<string>() { createResponse.GroupId }
        });
      return describeResponse.SecurityGroups;
    }



    //
    // Method to determine if a security group with the specified name
    // already exists in the VPC
    private static async Task<List<SecurityGroup>> FindSecurityGroups(
      IAmazonEC2 ec2Client, string groupName, string vpcID)
    {
      var request = new DescribeSecurityGroupsRequest();
      request.Filters.Add(new Filter{
        Name = "group-name",
        Values = new List<string>() { groupName }
      });
      if(!string.IsNullOrEmpty(vpcID))
        request.Filters.Add(new Filter{
          Name = "vpc-id",
          Values = new List<string>() { vpcID }
        });

      var response = await ec2Client.DescribeSecurityGroupsAsync(request);
      return response.SecurityGroups;
    }



    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: EC2CreateSecGroup -g <group-name> [-v <vpc-id>]" +
        "\n  -g, --group-name: The name you would like the new security group to have."
+
        "\n  -v, --vpc-id: The ID of a VPC to which the new security group will
belong." +
        "\n     If vpc-id isn't present, the security group will be" +
        "\n     for EC2-Classic (if your AWS account supports this)" +
        "\n     or will use the default VCP for EC2-VPC.");
```

```
    }
  }


  // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = =
  // 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);
    }
  }

}
```

セキュリティグループの更新

この例では、 を使用してセキュリティグループにルール AWS SDK for .NET 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
- 追加の考慮事項

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.");
    }
  }


  // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
 = = =
  // 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);
    }
  }

}
```

追加の考慮事項

- 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。の値は`IpProtocol`、IpPermission クラスの `IpProtocol`プロパティの説明にあります。

- 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」またはAmazon EC2 ユーザーガイド」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 キーペアの詳細については、Amazon EC2 ユーザーガイド」またはAmazon EC2 ユーザーガイド」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、 を使用してキーペア AWS SDK for .NET 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

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
{
  // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
 = = =
  // Class to create and store a key pair
  class Program
  {
    static async Task Main(string[] args)
    {
      // Create the EC2 client
      var ec2Client = new AmazonEC2Client();

      // Parse the command line and show help if necessary
      var parsedArgs = CommandLine.Parse(args);
      if(parsedArgs.Count == 0)
      {
        // In the case of no command-line arguments,
        // just show help and the existing key pairs
        PrintHelp();
        Console.WriteLine("\nNo arguments specified.");
        Console.Write(
          "Do you want to see a list of the existing key pairs? ((y) or n): ");
        string response = Console.ReadLine();
        if((string.IsNullOrEmpty(response)) || (response.ToLower() == "y"))
          await EnumerateKeyPairs(ec2Client);
        return;
      }

      // Get the application arguments from the parsed list
      string keyPairName =
        CommandLine.GetArgument(parsedArgs, null, "-k", "--keypair-name");
      string pemFileName =
        CommandLine.GetArgument(parsedArgs, null, "-p", "--pem-filename");
      if(string.IsNullOrEmpty(keyPairName))
        CommandLine.ErrorExit("\nNo key pair name specified." +
```

```
        "\nRun the command with no arguments to see help.");
    if(string.IsNullOrEmpty(pemFileName) || !pemFileName.EndsWith(".pem"))
      CommandLine.ErrorExit("\nThe PEM filename is missing or incorrect." +
        "\nRun the command with no arguments to see help.");

    // Create the key pair
    await CreateKeyPair(ec2Client, keyPairName, pemFileName);
    await EnumerateKeyPairs(ec2Client);
  }


  //
  // Method to create a key pair and save the key material in a PEM file
  private static async Task CreateKeyPair(
    IAmazonEC2 ec2Client, string keyPairName, string pemFileName)
  {
    // Create the key pair
    CreateKeyPairResponse response =
      await ec2Client.CreateKeyPairAsync(new CreateKeyPairRequest{
        KeyName = keyPairName
      });
    Console.WriteLine($"\nCreated new key pair: {response.KeyPair.KeyName}");

    // Save the private key in a PEM file
    using (var s = new FileStream(pemFileName, FileMode.Create))
    using (var writer = new StreamWriter(s))
    {
      writer.WriteLine(response.KeyPair.KeyMaterial);
    }
  }


  //
  // Method to show the key pairs that are available
  private static async Task EnumerateKeyPairs(IAmazonEC2 ec2Client)
  {
    DescribeKeyPairsResponse response = await ec2Client.DescribeKeyPairsAsync();
    Console.WriteLine("Available key pairs:");
    foreach (KeyPairInfo item in response.KeyPairs)
      Console.WriteLine($"  {item.KeyName}");
  }


  //
```

```
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: EC2CreateKeyPair -k <keypair-name> -p <pem-filename>" +
        "\n  -k, --keypair-name: The name you want to assign to the key pair." +
        "\n  -p, --pem-filename: The name of the PEM file to create, with a \".pem\"
 extension.");
    }
  }


  // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
 = = =
  // 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);
    }
  }
```

```
}
```

追加の考慮事項

- 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、 を使用してキーペア AWS SDK for .NET 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」またはAmazon EC2 ユーザーガイド」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」またはAmazon EC2 ユーザーガイド」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。

EC2 インスタンスは、VPC または EC2-Classic で起動できます (AWS アカウントでサポートされている場合) 。VPC 内の EC2 と EC2-Classic の詳細については、Amazon EC2 ユーザーガイド」またはAmazon EC2 ユーザーガイド」を参照してください。

> **Warning**
>
> 2022 年 8 月 15 日に、EC2-Classic の提供を終了し h ます。EC2-Classic は、VPC への移行をお勧めします。詳細については、Amazon EC2 EC2-Classic から VPC への移行Amazon EC2」を参照してください。ブログ記事「EC2-Classic Networking は販売終了になります — 準備方法はこちら」も参照してください。

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
- 追加の考慮事項
- (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が、 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。 CreateSubnetAsync

> **Note**
>
> AWS アカウントが EC2-Classic をサポートしていて、起動するインスタンスのタイプである場合、このパラメータは必要ありません。しかし、お使いのアカウントで EC2-Classic がサポートされておらず、このパラメータも指定されていない場合、新しいインスタンスはアカウントのデフォルトの VPC で起動します。

- 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「追加の考慮事項」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 については、Amazon EC2 ユーザーガイド」またはAmazon EC2 ユーザーガイド」を参照してください。例えば、Amazon EC2 ユーザーガイドAMIs Amazon EC2 について」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
{
  // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
 = = =
  // Class to launch an EC2 instance
  class Program
  {
    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
      string groupID =
        CommandLine.GetArgument(parsedArgs, null, "-g", "--group-id");
      string ami =
        CommandLine.GetArgument(parsedArgs, null, "-a", "--ami-id");
      string keyPairName =
        CommandLine.GetArgument(parsedArgs, null, "-k", "--keypair-name");
      string subnetID =
        CommandLine.GetArgument(parsedArgs, null, "-s", "--subnet-id");
      if(   (string.IsNullOrEmpty(groupID) || !groupID.StartsWith("sg-"))
         || (string.IsNullOrEmpty(ami) || !ami.StartsWith("ami-"))
         || (string.IsNullOrEmpty(keyPairName))
         || (!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);
    }
  }

}
```

追加の考慮事項

- EC2 インスタンスの状態を確認するときに、 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および MaxCountプロパティに使用できる値は、ターゲットアベイラビリティーゾーンと、インスタンスタイプで許可され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。このタイプについては、Amazon EC2 ユーザーガイド」またはAmazon EC2 ユーザーガイド」を参照してください。

- 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。これを行うには、Nameプロパティが IAM ロールの名前に設定されている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」またはAmazon EC2 ユーザーガイド」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 インスタンスに接続する」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 インスタンスへの接続」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
{
```

```
  // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = =
  // Class to get the Administrator password of a Windows EC2 instance
  class Program
  {
    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
      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.");
    }
  }

  // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = =
  // 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);
    }
  }

}
```

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、 を使用して EC2 インスタンス AWS SDK for .NET 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、 を使用して Amazon EC2 スポットインスタンス AWS SDK for .NET 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」または「Amazon EC2 ユーザーガイド」を参照してくださいAmazon EC2。

このチュートリアルの内容

このチュートリアルでは、 を使用して次の AWS SDK for .NET 操作を行います。

- 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

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。これらは、Amazon EC2 ユーザーガイド または Amazon EC2 ユーザーガイド で確認できます。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 については、Amazon EC2 ユーザーガイド」またはAmazon EC2 ユーザーガイド」を参照してください。例えば、Amazon EC2 ユーザーガイドAMIs Amazon EC2 について」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」または「Amazon ECAmazon EC2 ユーザーガイド」のスポット料金履歴を参照しています。

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、返されたSpotInstanceRequestオブジェクトSpotInstanceRequestIdのメンバーに含まれるスポットインスタンスリクエスト ID です。

> **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は、Amazon EC2 ユーザーガイド」またはAmazon EC2 ユーザーガイド」で確認できます。

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");
          }
        }
      }
    }
}
```

追加の考慮事項

- 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 Identity and Access Management、お客様が でユーザーとユーザーのアクセス許可 AWS を管理できるようにするウェブサービスである AWS SDK for .NET をサポートします AWS。

AWS Identity and Access Management (IAM) ユーザーは、 で作成するエンティティです AWS。エンティティは、 とやり取りする人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、専用ソフトウェアの開発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 Identity Center 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。APIsAWS SDK for .NET リファレンス」を参照してください (「Amazon」までスクロールIdentityManagement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 APIsは . AWSSDKIdentityManagement 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、環境とプロジェクトがセットアップされていることを必ず確認してください。また、「SDK の機能」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、 を使用して AWS SDK for .NET 、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
- 追加の考慮事項

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 AmazonIdentityManagement。モデル

  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
{
  // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
 = = =
  // Class to create an IAM policy with a given policy document
  class Program
  {
    private const int MaxArgs = 2;

    static async Task Main(string[] args)
    {
      // Parse the command line and show help if necessary
      var parsedArgs = CommandLine.Parse(args);
      if((parsedArgs.Count == 0) || (parsedArgs.Count > MaxArgs))
      {
        PrintHelp();
        return;
      }

      // Get the application arguments from the parsed list
```

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

```
  // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = =
  // 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);
    }
  }

}
```

追加の考慮事項

- 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。

```
{
```

```
  "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、 を使用してポリシードキュメント AWS SDK for .NET 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 AmazonIdentityManagement。モデル

  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、 を使用して Amazon EC2 インスタンスで IAM ロール AWS SDK for .NET 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、 が発行した認証情報を使用して暗号で署名 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」またはAmazon EC2 ユーザーガイド」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サービスのクライアントオブジェクトを構築すると AWS 、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に従うときは、 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で SDK 認証を設定します。

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
{
  // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
 = = =
  // Class to retrieve a text file from an S3 bucket and write it to a local file
  class Program
  {
    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
      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.");
    }
  }


  // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
 = = =
  // 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);
    }
  }

}
```

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 service] (AWS サービス) を選択し、[EC2] を見つけて選択して、[Next: Permissions] (次へ: アクセス許可) を選択します。
4. 「アクセス許可ポリシーをアタッチする」で、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

  Amazon EC2 ユーザーガイド」または「Amazon Amazon EC2」の指示に従ってインスタンスを起動します。

  先ほど作成した IAM ロールを選択できるように、ウィザードに従って少なくとも [Configure Instance Details] (インスタンスの詳細の設定) ページにアクセスする必要があります。

- 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「追加の考慮事項」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」またはAmazon EC2 ユーザーガイド」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。

これを行うには、Amazon EC2 ユーザーガイド」またはAmazon EC2 ユーザーガイド」の接続手順に従います。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」またはAmazon EC2 ユーザーガイド」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 を提供します。 Amazon S3 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 APIs は、以下の NuGet パッケージで提供されています。

- AWSSDK.S3
- Amazon.Extensions.S3.Encryption

前提条件

開始する前に、環境とプロジェクトがセットアップされていることを必ず確認してください。また、「SDK の機能」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、 を使用して Amazon S3 AWS SDK for .NET 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 Amazon S3デベロッパーガイドへのリンクでは、 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 用に作成されましたが、.NET Core AWS SDK for .NET を使用する の以降のバージョンでも実行できます。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 を使用したマルチパートアップロード (Amazon.S3.TransferTransferUtility)
- 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、それを使用して

クライアント側の暗号化用の 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」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
- 追加の考慮事項

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 AmazonKeyManagementService。モデル

  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
{
```

```
  // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = =
  // Class to store text in an encrypted S3 object.
  class Program
  {
    private const int MaxArgs = 3;

    public static async Task Main(string[] args)
    {
      // Parse the command line and show help if necessary
      var parsedArgs = CommandLine.Parse(args);
      if((parsedArgs.Count == 0) || (parsedArgs.Count > MaxArgs))
      {
        PrintHelp();
        return;
      }

      // Get the application arguments from the parsed list
      string bucketName =
        CommandLine.GetArgument(parsedArgs, null, "-b", "--bucket-name");
      string fileName =
        CommandLine.GetArgument(parsedArgs, null, "-f", "--file-name");
      string itemName =
        CommandLine.GetArgument(parsedArgs, null, "-i", "--item-name");
      if(string.IsNullOrEmpty(bucketName) || (string.IsNullOrEmpty(fileName)))
        CommandLine.ErrorExit(
          "\nOne or more of the required arguments is missing or incorrect." +
          "\nRun the command with no arguments to see help.");
      if(!File.Exists(fileName))
        CommandLine.ErrorExit($"\nThe given file {fileName} doesn't exist.");
      if(string.IsNullOrEmpty(itemName))
        itemName = Path.GetFileName(fileName);

      // Create a customer master key (CMK) and store the result
      CreateKeyResponse createKeyResponse =
        await new AmazonKeyManagementServiceClient().CreateKeyAsync(new
CreateKeyRequest());
      var kmsEncryptionContext = new Dictionary<string, string>();
      var kmsEncryptionMaterials = new EncryptionMaterialsV2(
        createKeyResponse.KeyMetadata.KeyId, KmsType.KmsContext, kmsEncryptionContext);

      // Create the object in the bucket, then display the content of the object
      var putObjectResponse =
```

```
        await CreateAndRetrieveObjectAsync(kmsEncryptionMaterials, bucketName,
fileName, itemName);
      Stream stream = putObjectResponse.ResponseStream;
      StreamReader reader = new StreamReader(stream);
      Console.WriteLine(reader.ReadToEnd());
    }


    //
    // Method to create and encrypt an object in an S3 bucket
    static async Task<GetObjectResponse> CreateAndRetrieveObjectAsync(
      EncryptionMaterialsV2 materials, string bucketName,
      string fileName, string itemName)
    {
      // CryptoStorageMode.ObjectMetadata is required for KMS EncryptionMaterials
      var config = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy)
      {
        StorageMode = CryptoStorageMode.ObjectMetadata
      };
      var s3EncClient = new AmazonS3EncryptionClientV2(config, materials);

      // Create, encrypt, and put the object
      await s3EncClient.PutObjectAsync(new PutObjectRequest
      {
        BucketName = bucketName,
        Key = itemName,
        ContentBody = File.ReadAllText(fileName)
      });

      // Get, decrypt, and return the object
      return await s3EncClient.GetObjectAsync(new GetObjectRequest
      {
        BucketName = bucketName,
        Key = itemName
      });
    }


    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: KmsS3Encryption -b <bucket-name> -f <file-name> [-i <item-name>]" +
```

```
        "\n  -b, --bucket-name: The name of an existing S3 bucket." +
        "\n  -f, --file-name: The name of a text file with content to encrypt and store
in S3." +
        "\n  -i, --item-name: The name you want to use for the item." +
        "\n      If item-name isn't given, file-name will be used.");
    }

  }

  // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = =
  // 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);
    }
  }

}
```

追加の考慮事項

- 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 クラスは、標準AmazonS3Clientクラスと同じインターフェイスを実装します。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。

- マスター AWS KMS キーとして キーを使用する利点の 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。その逆も同様です。

  > **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、AmazonS3EncryptionClientV2の クラス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 と AWS SDK for .NET バージョン 3.3 以前に基づく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。これは、アプリケーション、エンドユーザー、および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 を使用しますAmazonSimpleNotificationServiceClient。

```
// using Amazon.SimpleNotificationService;
// using Amazon.SimpleNotificationService.Model;

var client = new AmazonSimpleNotificationServiceClient();
var request = new ListTopicsRequest();
var response = new ListTopicsResponse();

do
{
  response = client.ListTopics(request);

  foreach (var topic in response.Topics)
  {
    Console.WriteLine("Topic: {0}", topic.TopicArn);

    var subs = client.ListSubscriptionsByTopic(
      new ListSubscriptionsByTopicRequest
      {
        TopicArn = topic.TopicArn
      });

    var ss = subs.Subscriptions;

    if (ss.Any())
    {
      Console.WriteLine("  Subscriptions:");

      foreach (var sub in ss)
      {
        Console.WriteLine("    {0}", sub.SubscriptionArn);
      }
    }
```

```
    var attrs = client.GetTopicAttributes(
      new GetTopicAttributesRequest
      {
        TopicArn = topic.TopicArn
      }).Attributes;

    if (attrs.Any())
    {
      Console.WriteLine("  Attributes:");

      foreach (var attr in attrs)
      {
        Console.WriteLine("    {0} = {1}", attr.Key, attr.Value);
      }
    }

    Console.WriteLine();
  }

  request.NextToken = response.NextToken;

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

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);
            }
        }
    }
}
```

コマンドラインからサンプルを構築して実行する方法については、「」の「完全な例」を参照してください GitHub。

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);
```

```
            }
        }
    }
}
```

コマンドラインからサンプルを構築して実行する方法については、「」の「完全な例」を参照してください GitHub。

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 をサポートしています。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 を提供します。 Amazon SQS 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 APIs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、環境とプロジェクトがセットアップされていることを必ず確認してください。また、「SDK の機能」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、 を使用して Amazon SQS キュー AWS SDK for .NET 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
- 追加の考慮事項

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
{
  // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
 = = =
  // Class to create a queue
  class Program
  {
```

```
    private const string MaxReceiveCount = "10";
    private const string ReceiveMessageWaitTime = "2";
    private const int MaxArgs = 3;

    static async Task Main(string[] args)
    {
      // Parse the command line and show help if necessary
      var parsedArgs = CommandLine.Parse(args);
      if(parsedArgs.Count > MaxArgs)
        CommandLine.ErrorExit(
          "\nToo many command-line arguments.\nRun the command with no arguments to see
 help.");

      // Create the Amazon SQS client
      var sqsClient = new AmazonSQSClient();

      // In the case of no command-line arguments, just show help and the existing
 queues
      if(parsedArgs.Count == 0)
      {
        PrintHelp();
        Console.WriteLine("\nNo arguments specified.");
        Console.Write("Do you want to see a list of the existing queues? ((y) or n):
 ");
        string response = Console.ReadLine();
        if((string.IsNullOrEmpty(response)) || (response.ToLower() == "y"))
          await ShowQueues(sqsClient);
        return;
      }

      // Get the application arguments from the parsed list
      string queueName =
        CommandLine.GetArgument(parsedArgs, null, "-q", "--queue-name");
      string deadLetterQueueUrl =
        CommandLine.GetArgument(parsedArgs, null, "-d", "--dead-letter-queue");
      string maxReceiveCount =
        CommandLine.GetArgument(parsedArgs, MaxReceiveCount, "-m", "--max-receive-
count");
      string receiveWaitTime =
        CommandLine.GetArgument(parsedArgs, ReceiveMessageWaitTime, "-w", "--wait-
time");

      if(string.IsNullOrEmpty(queueName))
        CommandLine.ErrorExit(
```

```
          "\nYou must supply a queue name.\nRun the command with no arguments to see
help.");

      // If a dead-letter queue wasn't given, create one
      if(string.IsNullOrEmpty(deadLetterQueueUrl))
      {
        Console.WriteLine("\nNo dead-letter queue was specified. Creating one...");
        deadLetterQueueUrl = await CreateQueue(sqsClient, queueName + "__dlq");
        Console.WriteLine($"Your new dead-letter queue:");
        await ShowAllAttributes(sqsClient, deadLetterQueueUrl);
      }

      // Create the message queue
      string messageQueueUrl = await CreateQueue(
        sqsClient, queueName, deadLetterQueueUrl, maxReceiveCount, receiveWaitTime);
      Console.WriteLine($"Your new message queue:");
      await ShowAllAttributes(sqsClient, messageQueueUrl);
    }


    //
    // Method to show a list of the existing queues
    private static async Task ShowQueues(IAmazonSQS sqsClient)
    {
      ListQueuesResponse responseList = await sqsClient.ListQueuesAsync("");
      Console.WriteLine();
      foreach(string qUrl in responseList.QueueUrls)
      {
        // Get and show all attributes. Could also get a subset.
        await ShowAllAttributes(sqsClient, qUrl);
      }
    }


    //
    // Method to create a queue. Returns the queue URL.
    private static async Task<string> CreateQueue(
      IAmazonSQS sqsClient, string qName, string deadLetterQueueUrl=null,
      string maxReceiveCount=null, string receiveWaitTime=null)
    {
      var attrs = new Dictionary<string, string>();

      // If a dead-letter queue is given, create a message queue
      if(!string.IsNullOrEmpty(deadLetterQueueUrl))
```

```
      {
        attrs.Add(QueueAttributeName.ReceiveMessageWaitTimeSeconds, receiveWaitTime);
        attrs.Add(QueueAttributeName.RedrivePolicy,
          $"{{\"deadLetterTargetArn\":\"{await GetQueueArn(sqsClient,
deadLetterQueueUrl)}\"," +
          $"\"maxReceiveCount\":\"{maxReceiveCount}\"}}");
        // Add other attributes for the message queue such as VisibilityTimeout
      }

      // If no dead-letter queue is given, create one of those instead
      //else
      //{
      //  // Add attributes for the dead-letter queue as needed
      //  attrs.Add();
      //}

      // Create the queue
      CreateQueueResponse responseCreate = await sqsClient.CreateQueueAsync(
          new CreateQueueRequest{QueueName = qName, Attributes = attrs});
      return responseCreate.QueueUrl;
    }



    //
    // Method to get the ARN of a queue
    private static async Task<string> GetQueueArn(IAmazonSQS sqsClient, string qUrl)
    {
      GetQueueAttributesResponse responseGetAtt = await
sqsClient.GetQueueAttributesAsync(
        qUrl, new List<string>{QueueAttributeName.QueueArn});
      return responseGetAtt.QueueARN;
    }



    //
    // Method to show all attributes of a queue
    private static async Task ShowAllAttributes(IAmazonSQS sqsClient, string qUrl)
    {
      var attributes = new List<string>{ QueueAttributeName.All };
      GetQueueAttributesResponse responseGetAtt =
        await sqsClient.GetQueueAttributesAsync(qUrl, attributes);
      Console.WriteLine($"Queue: {qUrl}");
      foreach(var att in responseGetAtt.Attributes)
        Console.WriteLine($"\t{att.Key}: {att.Value}");
```

```
    }


    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
      "\nUsage: SQSCreateQueue -q <queue-name> [-d <dead-letter-queue>]" +
        " [-m <max-receive-count>] [-w <wait-time>]" +
      "\n  -q, --queue-name: The name of the queue you want to create." +
      "\n  -d, --dead-letter-queue: The URL of an existing queue to be used as the
dead-letter queue."+
      "\n      If this argument isn't supplied, a new dead-letter queue will be
created." +
      "\n  -m, --max-receive-count: The value for maxReceiveCount in the RedrivePolicy
of the queue." +
      $"\n      Default is {MaxReceiveCount}." +
      "\n  -w, --wait-time: The value for ReceiveMessageWaitTimeSeconds of the queue
for long polling." +
      $"\n      Default is {ReceiveMessageWaitTime}.");
    }
 }


 // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = =
 // 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);
    }
  }

}
```

追加の考慮事項

- 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CreateQueueRequestの または Amazon Simple Queue Service API リファレンスSetQueueAttributesの を参照してください。 https://docs.aws.amazon.com/AWSSimpleQueueService/latest/APIReference/

- 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、 を使用して Amazon SQS キュー AWS SDK for .NET 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

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
{
  // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
 = = =
  // Class to update a queue
  class Program
  {
    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.");
```

```
    }
  }


  // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = =
  // 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);
    }
  }

}
```

追加の考慮事項

- 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、 を使用して Amazon SQS キュー AWS SDK for .NET 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
- 追加の考慮事項

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
{
  // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
 = = =
  // Class to update a queue
  class Program
  {
```

```
    private const int TimeToWait = 60;

    static async Task Main(string[] args)
    {
      // Create the Amazon SQS client
      var sqsClient = new AmazonSQSClient();

      // If no command-line arguments, just show a list of the queues
      if(args.Length == 0)
      {
        Console.WriteLine("\nUsage: SQSCreateQueue queue_url");
        Console.WriteLine("   queue_url - The URL of the queue you want to delete.");
        Console.WriteLine("\nNo arguments specified.");
        Console.Write("Do you want to see a list of the existing queues? ((y) or n):
");
        var response = Console.ReadLine();
        if((string.IsNullOrEmpty(response)) || (response.ToLower() == "y"))
          await ListQueues(sqsClient);
        return;
      }

      // If given a queue URL, delete that queue
      if(args[0].StartsWith("https://sqs."))
      {
        // Delete the queue
        await DeleteQueue(sqsClient, args[0]);
        // Wait for a little while because it takes a while for the queue to disappear
        await Wait(sqsClient, TimeToWait, args[0]);
        // Show a list of the remaining queues
        await ListQueues(sqsClient);
      }
      else
      {
        Console.WriteLine("The command-line argument isn't a queue URL:");
        Console.WriteLine($"{args[0]}");
      }
    }


    //
    // Method to delete an SQS queue
    private static async Task DeleteQueue(IAmazonSQS sqsClient, string qUrl)
    {
      Console.WriteLine($"Deleting queue {qUrl}...");
```

```
      await sqsClient.DeleteQueueAsync(qUrl);
      Console.WriteLine($"Queue {qUrl} has been deleted.");
    }


    //
    // Method to wait up to a given number of seconds
    private static async Task Wait(
      IAmazonSQS sqsClient, int numSeconds, string qUrl)
    {
      Console.WriteLine($"Waiting for up to {numSeconds} seconds.");
      Console.WriteLine("Press any key to stop waiting. (Response might be slightly
delayed.)");
      for(int i=0; i<numSeconds; i++)
      {
        Console.Write(".");
        Thread.Sleep(1000);
        if(Console.KeyAvailable) break;

        // Check to see if the queue is gone yet
        var found = false;
        ListQueuesResponse responseList = await sqsClient.ListQueuesAsync("");
        foreach(var url in responseList.QueueUrls)
        {
          if(url == qUrl)
          {
            found = true;
            break;
          }
        }
        if(!found) break;
      }
    }


    //
    // Method to show a list of the existing queues
    private static async Task ListQueues(IAmazonSQS sqsClient)
    {
      ListQueuesResponse responseList = await sqsClient.ListQueuesAsync("");
      Console.WriteLine("\nList of queues:");
      foreach(var qUrl in responseList.QueueUrls)
        Console.WriteLine($"- {qUrl}");
    }
```

```
    }
}
```

追加の考慮事項

- 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。Amazon SQS 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。 Amazon SQS 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
- 追加の考慮事項

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
{
  // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
 = = =
  // Class to send messages to a queue
  class Program
  {
    // Some example messages to send to the queue
    private const string JsonMessage = "{\"product\":[{\"name\":\"Product A\",\"price
\": \"32\"},{\"name\": \"Product B\",\"price\": \"27\"}]}";
    private const string XmlMessage = "<products><product name=\"Product A\" price=
\"32\" /><product name=\"Product B\" price=\"27\" /></products>";
    private const string CustomMessage = "||product|Product A|32||product|Product B|
27||";
    private const string TextMessage = "Just a plain text message.";

    static async Task Main(string[] args)
    {
      // Do some checks on the command-line
      if(args.Length == 0)
      {
        Console.WriteLine("\nUsage: SQSSendMessages queue_url");
        Console.WriteLine("   queue_url - The URL of an existing SQS queue.");
        return;
      }
      if(!args[0].StartsWith("https://sqs."))
      {
        Console.WriteLine("\nThe command-line argument isn't a queue URL:");
        Console.WriteLine($"{args[0]}");
        return;
      }

      // Create the Amazon SQS client
      var sqsClient = new AmazonSQSClient();

      // (could verify that the queue exists)
      // Send some example messages to the given queue
      // A single message
      await SendMessage(sqsClient, args[0], JsonMessage);

      // A batch of messages
      var batchMessages = new List<SendMessageBatchRequestEntry>{
        new SendMessageBatchRequestEntry("xmlMsg", XmlMessage),
        new SendMessageBatchRequestEntry("customeMsg", CustomMessage),
```

```
        new SendMessageBatchRequestEntry("textMsg", TextMessage)};
      await SendMessageBatch(sqsClient, args[0], batchMessages);

      // Let the user send their own messages or quit
      await InteractWithUser(sqsClient, args[0]);

      // Delete all messages that are still in the queue
      await DeleteAllMessages(sqsClient, args[0]);
    }



    //
    // Method to put a message on a queue
    // Could be expanded to include message attributes, etc., in a SendMessageRequest
    private static async Task SendMessage(
      IAmazonSQS sqsClient, string qUrl, string messageBody)
    {
      SendMessageResponse responseSendMsg =
        await sqsClient.SendMessageAsync(qUrl, messageBody);
      Console.WriteLine($"Message added to queue\n  {qUrl}");
      Console.WriteLine($"HttpStatusCode: {responseSendMsg.HttpStatusCode}");
    }



    //
    // Method to put a batch of messages on a queue
    // Could be expanded to include message attributes, etc.,
    // in the SendMessageBatchRequestEntry objects
    private static async Task SendMessageBatch(
      IAmazonSQS sqsClient, string qUrl, List<SendMessageBatchRequestEntry> messages)
    {
      Console.WriteLine($"\nSending a batch of messages to queue\n  {qUrl}");
      SendMessageBatchResponse responseSendBatch =
        await sqsClient.SendMessageBatchAsync(qUrl, messages);
      // Could test responseSendBatch.Failed here
      foreach(SendMessageBatchResultEntry entry in responseSendBatch.Successful)
        Console.WriteLine($"Message {entry.Id} successfully queued.");
    }



    //
    // Method to get input from the user
    // They can provide messages to put in the queue or exit the application
    private static async Task InteractWithUser(IAmazonSQS sqsClient, string qUrl)
```

```
    {
      string response;
      while (true)
      {
        // Get the user's input
        Console.WriteLine("\nType a message for the queue or \"exit\" to quit:");
        response = Console.ReadLine();
        if(response.ToLower() == "exit") break;

        // Put the user's message in the queue
        await SendMessage(sqsClient, qUrl, response);
      }
    }


    //
    // Method to delete all messages from the queue
    private static async Task DeleteAllMessages(IAmazonSQS sqsClient, string qUrl)
    {
      Console.WriteLine($"\nPurging messages from queue\n  {qUrl}...");
      PurgeQueueResponse responsePurge = await sqsClient.PurgeQueueAsync(qUrl);
      Console.WriteLine($"HttpStatusCode: {responsePurge.HttpStatusCode}");
    }
  }
}
```

追加の考慮事項

- 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。Amazon SQS 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。 Amazon SQS 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
- 追加の考慮事項

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);
    }
  }
}
```

追加の考慮事項

- 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。

コンピュートサービスでAWS Lambdaを使用する

AWS SDK for .NET は AWS Lambda をサポートしているため、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 は、AWS Lambda 用の API を提供しています。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 .NETAPI 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、環境とプロジェクトがセットアップされていることを必ず確認してください。また、「SDK の機能」の情報を確認してください。

トピック

トピック

- アノテーションを使用してAWS Lambda関数を記述する

アノテーションを使用して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。

AWS SDK for .NET の高レベルライブラリとフレームワーク

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。

AWS SDK for .NET を初めて使用する場合は、まず クイックツアーをする トピックを確認する必要があるかもしれません。SDK についてわかりやすく説明しています。

開始する前に、環境とプロジェクトがセットアップされていることを必ず確認してください。また、「SDK の機能」の情報を確認してください。

トピック

- AWS .NET 用の Message Processing Framework

AWS .NET 用の Message Processing Framework

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。

AWS Message Processing Framework for .NET は、Amazon Simple Queue Service (SQS)、Amazon Simple Notification Service (SNS)、Amazon 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 EventBridge。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、低レベルの API コール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。

Message Processing Framework 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 および にイベントを発行します EventBridge。
- バックグラウンドサービスで通常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処理できないようにメッセージが処理されている間の可視性タイムアウトの管理が含まれます。
- 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。
- OpenTelemetry ログ記録用の 。

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、以下にリストされているトピックを参照してください。

開始する前に、環境とプロジェクトがセットアップされている ことを必ず確認してください。また、「SDK の機能」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 リポジトリ内の README ファイル https://github.com/awslabs/aws-dotnet-messaging
- Microsoft からの .NET 依存関係インジェクション。
- Microsoft の .NET Generic Host。

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 AWS Processing Framework for .NET でメッセージを消費する
- .NET 用の AWS Message Processing Framework での FIFO の使用
- AWS Message Processing Framework for .NET のログ記録とオープンテレメトリ
- 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

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。

開始する前に、環境とプロジェクトがセットアップされている ことを必ず確認してください。また、「SDK の機能」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、EventBridgeそれぞれのデベロッパーガイドの SQS 、SNS 、および のセキュリティトピックを参照してください。
- 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ます。アプリケーションの起動時にフレームワークを設定するには、 AddAWSMessageBusを呼び出して DI コンテナに追加します。

```
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ツールチェーンのインストールと設定。
- 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で SDK 認証を設定します「」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 に対して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。Message Processing Framework for AWS .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 Identity Center に を使用している場合は、必ず AWSSDK.SSOと を追加してください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ます。これにより、Swagger UI でブラウザウィンドウが開き、API の探索とテストが可能になります。

```
dotnet watch run <queue URL created earlier>
```

6. /greeting エンドポイントを開き、 を試す を選択します。
7. メッセージの 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。Message Processing Framework for AWS .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 Identity Center に を使用している場合は、必ず AWSSDK.SSOと を追加してください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ます。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>
```

起動後すぐに、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

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。

Message AWS Processing Framework for .NET では、1 つ以上のメッセージタイプの公開、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その両方の実行がサポートされています。

次のコードは、異なるメッセージタイプを異なる AWS サービスに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、上記で設定し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、そのタイプに基づいて、メッセージのルーティング先を決定します。

次の例では、ASP.NET MVC コントローラーはユーザーから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、上記で設定し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、このメカニズムは、プロジェクト間でメッセージオブジェクトの定義を共有するとうまく機能します。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、またはメッセージを他のフレームワークやプログラミング言語と交換する場合は、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。汎用 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 のサービス固有のパブリッシャーも提供します EventBridge。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。

例えば、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ChatMessage例を再度示していますが、現在は を使用して SQS 固有のオプションISQSPublisher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 では EventBridge、IEventBridgePublisherそれぞれ ISNSPublisherと を使用して同じことができます。

```
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 クライアントをオーバーライド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 AWS Processing Framework for .NET でメッセージを消費する

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。

Message AWS Processing Framework for .NET では、フレームワークまたはメッセージングサービスのいずれか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消費できます。そのいくつかを以下に示します。

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ChatMessageメッセージハンドラーの例を示しています。

```
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留まり、再度処理するか、設定されている場合はデッドレターキューに移動できます。

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 を呼び出すと、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 メッセージポーラーの設定

SQS メッセージポーラーは、 を呼び出すSQSMessagePollerOptionsときに で設定できますAddSQSPoller。

- MaxNumberOfConcurrentMessages - キューから同時に処理でき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時間 (秒単位) 。メッセージが利用可能な場合、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。

メッセージの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。1 つの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留まりますが、他のコンシューマーからは複数回処理されないように非表示になります。再び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。

このフレームワークは、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 - 受信したメッセージの秒単位の期間は、後続の取得リクエストでは非表示になります。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この数秒以内に期限切れになると、フレームワークは可視性タイムアウトを (別の`VisibilityTimeout`秒) 延長します。デフォルト値は 5 です。
- `VisibilityTimeoutExtensionHeartbeatInterval` - フレームワークが期限切れ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。再び表示されてから 5 秒以内のメッセージの場合、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。

.NET 用の AWS Message Processing Framework での FIFO の使用

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。

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 固有のメッセージ発行者に設定できます。

```
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、フレームワークは特定のメッセージグループ内のメッセージをReceiveMessages、呼び出しごとに受信した順序で処理します。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 のログ記録とオープンテレメトリ

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。

Message AWS Processing Framework for .NET は、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 OpenTelemetry に記録するために 用に計測されています。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**
>
> ログ記録に関連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

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。

Message AWS 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、サービスで定義されているMessageオブジェクトを操作します。
2. SQS リクエストとレスポンス内では、フレームワークは MessageBody要素 (または Amazon SNSMessage の場合は 、Amazon Detailの場合は EventBridge) を JSON 形式の CloudEventに設定します。これには、メッセージを処理する際に 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、メッセージがハンドラーにルーティングされ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 経由で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オーバーライドできます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、 から発行する場合は関数名 AWS Lambda、Amazon ECS で公開する場合はクラスター名とタスク ARN、Amazon EC2 で公開する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" は UTC DateTime で現在の に設定されます。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オーバーライドできます。
- "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 の MessageBusBuilder では、メッセージのシリアル化と逆シリアル化System.Text.Json.JsonSerializerOptionsに使用する を設定できます。
  - フレームワークが構築した後、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の を使用してその属性を実装ISerializationCallbackして登録できますMessageBusBuilder。

AWS Message Processing Framework for .NET のセキュリティ

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。

Message AWS Processing Framework for .NET は、 との通信 AWS SDK for .NET に に依存しています 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によって送信されたデータメッセージをログに記録しません。デバッグ目的でこの機能を有効にする場合は、次のようにメッセージバスEnableDataMessageLogging()で 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が見つかった場合は、セキュリティポリシーを参照して情報を報告してください。

スタックとアプリケーションを使用するための AWS OpsWorks のプログラミング

> **Warning**
>
> AWS OpsWorks はサポート終了間近で、新規のお客様は受け付けていません。既存のお客様は、使用しているサービスに応じて 2024 年 3 月または 5 月まではこれによる影響はなく、その時点でサービスは利用できなくなります。この移行に備えるため、既存のお客様は、可能な限り早急に他のソリューションに移行することをお勧めします。詳細については、OpsWorks 製品ページを参照してください。

AWS SDK for .NET は AWS OpsWorks をサポートします。これは、スタックとアプリケーションを作成および管理するためのシンプルで柔軟性のある方法を提供します。AWS OpsWorks を使用すると、AWS リソースのプロビジョニング、AWS リソースの設定の管理、AWS リソースへのアプリケーションのデプロイ、および AWS リソースの状態のモニタリングを行うことができます。詳細については、OpsWorks 製品ページとAWS OpsWorks ユーザーガイドを参照してください。

API

AWS SDK for .NET は、AWS OpsWorks 用の API を提供しています。API を使用すると、レイヤー、インスタンス、アプリを含んでいるスタックなどの AWS OpsWorks 機能を使用できます。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.OpsWorks」までスクロールします)。

AWS OpsWorks API は、AWSSDK.OpsWorks 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、環境とプロジェクトがセットアップされていることを必ず確認してください。また、「SDK の機能」の情報を確認してください。

他の AWS サービスと設定のサポート

AWS SDK for .NET は、前のセクションで説明したサービス以外の AWS サービスもサポートしています。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。

個別の AWS のサービス用の名前空間に加えて、AWS SDK for .NET では以下の API も提供されています。

エリア	説明	リソース
AWS サポート	AWS サポートケースおよび 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。

AWS SDK for .NET コード例

このトピックのコード例は、 AWS SDK for .NET で を使用する方法を示しています AWS。

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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、特定のタスクを実行する方法を示すコード例です。

クロスサービスの例は、複数の AWS のサービスで動作するサンプルアプリケーションです。

例

- を使用したアクションとシナリオ AWS SDK for .NET
- を使用したクロスサービスの例 AWS SDK for .NET

を使用したアクションとシナリオ AWS SDK for .NET

次のコード例は、 AWS SDK for .NET で 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 のサービス。

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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、特定のタスクを実行する方法を示すコード例です。

サービス

- を使用した ACM の例 AWS SDK for .NET
- を使用した Aurora の例 AWS SDK for .NET
- を使用した Auto Scaling の例 AWS SDK for .NET
- を使用した Amazon Bedrock の例 AWS SDK for .NET
- を使用した Amazon Bedrock ランタイムの例 AWS SDK for .NET
- AWS CloudFormation を使用した の例 AWS SDK for .NET
- CloudWatch を使用した の例 AWS SDK for .NET

- CloudWatch を使用したログ記録の例 AWS SDK for .NET
- を使用した Amazon Cognito ID プロバイダーの例 AWS SDK for .NET
- を使用した Amazon Comprehend の例 AWS SDK for .NET
- を使用した DynamoDB の例 AWS SDK for .NET
- を使用した Amazon EC2 の例 AWS SDK for .NET
- を使用した Amazon ECS の例 AWS SDK for .NET
- を使用した Elastic Load Balancing - バージョン 2 の例 AWS SDK for .NET
- EventBridge を使用した の例 AWS SDK for .NET
- AWS Glue を使用した の例 AWS SDK for .NET
- を使用した IAM の例 AWS SDK for .NET
- を使用した Amazon Keyspaces の例 AWS SDK for .NET
- を使用した Kinesis の例 AWS SDK for .NET
- AWS KMS を使用した の例 AWS SDK for .NET
- を使用した Lambda の例 AWS SDK for .NET
- MediaConvert を使用した の例 AWS SDK for .NET
- を使用した Organizations の例 AWS SDK for .NET
- を使用した Amazon Pinpoint の例 AWS SDK for .NET
- を使用した Amazon Polly の例 AWS SDK for .NET
- を使用した Amazon RDS の例 AWS SDK for .NET
- を使用した Amazon Rekognition の例 AWS SDK for .NET
- を使用した Route 53 ドメイン登録の例 AWS SDK for .NET
- を使用した Amazon S3 の例 AWS SDK for .NET
- を使用した S3 Glacier の例 AWS SDK for .NET
- SageMaker を使用した の例 AWS SDK for .NET
- を使用した Secrets Manager の例 AWS SDK for .NET
- を使用した Amazon SES の例 AWS SDK for .NET
- を使用した Amazon SES API v2 の例 AWS SDK for .NET
- を使用した Amazon SNS の例 AWS SDK for .NET
- を使用した Amazon SQS の例 AWS SDK for .NET
- を使用した Step Functions の例 AWS SDK for .NET

- AWS STS を使用した の例 AWS SDK for .NET
- AWS Support を使用した の例 AWS SDK for .NET
- を使用した Amazon Transcribe の例 AWS SDK for .NET
- を使用した Amazon Translate の例 AWS SDK for .NET

を使用した ACM の例 AWS SDK for .NET

次のコード例は、ACM AWS SDK for .NET で 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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。ここでは GitHub、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

- アクション

アクション

`DescribeCertificate`

次の例は、`DescribeCertificate` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
using System;
```

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

namespace DescribeCertificate
{
    class DescribeCertificate
    {
        // The following example retrieves and displays the metadata for a
        // certificate using the AWS Certificate Manager (ACM) service.

        // Specify your AWS Region (an example Region is shown).
        private static readonly RegionEndpoint ACMRegion = RegionEndpoint.USEast1;
        private static AmazonCertificateManagerClient _client;

        static void Main(string[] args)
        {
            _client = new
 Amazon.CertificateManager.AmazonCertificateManagerClient(ACMRegion);

            var describeCertificateReq = new DescribeCertificateRequest();
            // The ARN used here is just an example. Replace it with the ARN of
            // a certificate that exists on your account.
            describeCertificateReq.CertificateArn =
                "arn:aws:acm:us-
east-1:123456789012:certificate/8cfd7dae-9b6a-2d07-92bc-1c309EXAMPLE";

            var certificateDetailResp =
                DescribeCertificateResponseAsync(client: _client, request:
 describeCertificateReq);
            var certificateDetail = certificateDetailResp.Result.Certificate;

            if (certificateDetail is not null)
            {
                DisplayCertificateDetails(certificateDetail);
            }
        }

        /// <summary>
        /// Displays detailed metadata about a certificate retrieved
        /// using the ACM service.
        /// </summary>
        /// <param name="certificateDetail">The object that contains details
```

```
        /// returned from the call to DescribeCertificateAsync.</param>
        static void DisplayCertificateDetails(CertificateDetail certificateDetail)
        {
            Console.WriteLine("\nCertificate Details: ");
            Console.WriteLine($"Certificate Domain:
{certificateDetail.DomainName}");
            Console.WriteLine($"Certificate Arn:
{certificateDetail.CertificateArn}");
            Console.WriteLine($"Certificate Subject: {certificateDetail.Subject}");
            Console.WriteLine($"Certificate Status: {certificateDetail.Status}");
            foreach (var san in certificateDetail.SubjectAlternativeNames)
            {
                Console.WriteLine($"Certificate SubjectAlternativeName: {san}");
            }
        }

        /// <summary>
        /// Retrieves the metadata associated with the ACM service certificate.
        /// </summary>
        /// <param name="client">An AmazonCertificateManagerClient object
        /// used to call DescribeCertificateResponse.</param>
        /// <param name="request">The DescribeCertificateRequest object that
        /// will be passed to the method call.</param>
        /// <returns></returns>
        static async Task<DescribeCertificateResponse>
DescribeCertificateResponseAsync(
            AmazonCertificateManagerClient client, DescribeCertificateRequest
request)
        {
            var response = new DescribeCertificateResponse();

            try
            {
                response = await client.DescribeCertificateAsync(request);
            }
            catch (InvalidArnException)
            {
                Console.WriteLine($"Error: The ARN specified is invalid.");
            }
            catch (ResourceNotFoundException)
            {
                Console.WriteLine($"Error: The specified certificate could not be
found.");
            }
```

```
            return response;
        }
    }

}
```

- API の詳細については、「 API リファレンスDescribeCertificate」の「」を参照してください。 AWS SDK for .NET

`ListCertificates`

次の例は、`ListCertificates` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

namespace ListCertificates
{
    // The following example retrieves and displays a list of the
    // certificates defined for the default account using the AWS
    // Certificate Manager (ACM) service.
    class ListCertificates
    {
        // Specify your AWS Region (an example Region is shown).

        private static readonly RegionEndpoint ACMRegion = RegionEndpoint.USEast1;
```

```
        private static AmazonCertificateManagerClient _client;

        static void Main(string[] args)
        {
            _client = new AmazonCertificateManagerClient(ACMRegion);
            var certificateList = ListCertificatesResponseAsync(client: _client);

            Console.WriteLine("Certificate Summary List\n");

            foreach (var certificate in
 certificateList.Result.CertificateSummaryList)
            {
                Console.WriteLine($"Certificate Domain: {certificate.DomainName}");
                Console.WriteLine($"Certificate ARN:
 {certificate.CertificateArn}\n");
            }
        }

        /// <summary>
        /// Retrieves a list of the certificates defined in this Region.
        /// </summary>
        /// <param name="client">The ACM client object passed to the
        /// ListCertificateResAsync method call.</param>
        /// <param name="request"></param>
        /// <returns>The ListCertificatesResponse.</returns>
        static async Task<ListCertificatesResponse> ListCertificatesResponseAsync(
            AmazonCertificateManagerClient client)
        {
            var request = new ListCertificatesRequest();

            var response = await client.ListCertificatesAsync(request);
            return response;
        }
    }
}
```

- API の詳細については、「 API リファレンスListCertificates」の「」を参照してください。 AWS SDK for .NET

を使用した Aurora の例 AWS SDK for .NET

次のコード例は、Aurora AWS SDK for .NET で 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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。ここでは GitHub、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。

開始方法

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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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($"Hello Amazon RDS Aurora! Let's list some clusters in
 this account:");
        foreach (var cluster in response.DBClusters)
        {
            Console.WriteLine($"\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。

トピック

- アクション
- シナリオ

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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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、「 API リファレンスCreateDBClusterParameterGroup」を参照してください。 AWS SDK for .NET

`CreateDBClusterSnapshot`

次の例は、`CreateDBClusterSnapshot` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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、「 API リファレンスCreateDBClusterSnapshot」を参照してください。 AWS SDK for .NET

CreateDBInstance

次の例は、CreateDBInstance を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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、「 API リファレンスDeleteDBClusterParameterGroup」を参照してください。 AWS SDK for .NET

DeleteDBInstance

次の例は、DeleteDBInstance を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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、「 API DescribeDBClusterParameterGroupsAWS SDK for .NET 」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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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、「 API DescribeDBClusterParametersAWS SDK for .NET 」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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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、「 API DescribeDBClusterSnapshotsAWS SDK for .NET 」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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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);
            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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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、「 API DescribeDBEngineVersionsAWS SDK for .NET 」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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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、「 API リファレンス」のDescribeOrderable「DBInstanceOptions」を参照してください。 AWS SDK for .NET

ModifyDBClusterParameterGroup

次の例は、ModifyDBClusterParameterGroup を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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 && 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、「 API リファレンスModifyDBClusterParameterGroup」を参照してください。 AWS SDK for .NET

シナリオ

DB クラスターの開始方法

次のコードサンプルは、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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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の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 && 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);
            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

を使用した Auto Scaling の例 AWS SDK for .NET

次のコード例は、Auto Scaling AWS SDK for .NET で 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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。ここでは GitHub、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。

開始方法

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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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();

        response.AutoScalingGroups.ForEach(autoScalingGroup =>
        {

 Console.WriteLine($"{autoScalingGroup.AutoScalingGroupName}\t{autoScalingGroup.Availability
        });

        if (response.AutoScalingGroups.Count == 0)
        {
            Console.WriteLine("Sorry, you don't have any Amazon EC2 Auto Scaling
 groups.");
        }
    }
}
```

- API の詳細については、「 API リファレンスDescribeAutoScalingGroups」の「」を参照してください。 AWS SDK for .NET

トピック

- アクション
- シナリオ

アクション

`AttachLoadBalancerTargetGroups`

次の例は、AttachLoadBalancerTargetGroups を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Attaches an Elastic Load Balancing (ELB) target group to this EC2 Auto
 Scaling group.
    /// The
    /// </summary>
    /// <param name="autoScalingGroupName">The name of the Auto Scaling group.</
param>
    /// <param name="targetGroupArn">The Arn for the target group.</param>
    /// <returns>Async task.</returns>
    public async Task AttachLoadBalancerToGroup(string autoScalingGroupName, string
 targetGroupArn)
    {
        await _amazonAutoScaling.AttachLoadBalancerTargetGroupsAsync(
            new AttachLoadBalancerTargetGroupsRequest()
            {
                AutoScalingGroupName = autoScalingGroupName,
                TargetGroupARNs = new List<string>() { targetGroupArn }
            });
    }
```

- API の詳細については、「 API リファレンスAttachLoadBalancerTargetGroups」の「」を参照してください。 AWS SDK for .NET

`CreateAutoScalingGroup`

次の例は、CreateAutoScalingGroup を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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
    };
```

```
        var response = await
_amazonAutoScaling.CreateAutoScalingGroupAsync(request);
        Console.WriteLine($"{groupName} Auto Scaling Group created");
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスCreateAutoScalingGroup」の「」を参照してください。 AWS SDK for .NET

`DeleteAutoScalingGroup`

次の例は、`DeleteAutoScalingGroup` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Auto Scaling グループの最小サイズをゼロに更新し、グループ内のすべてのインスタンスを終了して、グループを削除します。

```
    /// <summary>
    /// Try to terminate an instance by its Id.
    /// </summary>
    /// <param name="instanceId">The Id of the instance to terminate.</param>
    /// <returns>Async task.</returns>
    public async Task TryTerminateInstanceById(string instanceId)
    {
        var stopping = false;
        Console.WriteLine($"Stopping {instanceId}...");
        while (!stopping)
        {
            try
            {
                await _amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(
                    new TerminateInstanceInAutoScalingGroupRequest()
                    {
```

```
                        InstanceId = instanceId,
                        ShouldDecrementDesiredCapacity = false
                    });
                stopping = true;
            }
            catch (ScalingActivityInProgressException)
            {
                Console.WriteLine($"Scaling activity in progress for {instanceId}.
Waiting...");
                Thread.Sleep(10000);
            }
        }
    }

    /// <summary>
    /// Tries to delete the EC2 Auto Scaling group. If the group is in use or in
progress,
    /// waits and retries until the group is successfully deleted.
    /// </summary>
    /// <param name="groupName">The name of the group to try to delete.</param>
    /// <returns>Async task.</returns>
    public async Task TryDeleteGroupByName(string groupName)
    {
        var stopped = false;
        while (!stopped)
        {
            try
            {
                await _amazonAutoScaling.DeleteAutoScalingGroupAsync(
                    new DeleteAutoScalingGroupRequest()
                    {
                        AutoScalingGroupName = groupName
                    });
                stopped = true;
            }
            catch (Exception e)
                when ((e is ScalingActivityInProgressException)
                      || (e is Amazon.AutoScaling.Model.ResourceInUseException))
            {
                Console.WriteLine($"Some instances are still running. Waiting...");
                Thread.Sleep(10000);
            }
        }
    }
```

```
    /// <summary>
    /// Terminate instances and delete the Auto Scaling group by name.
    /// </summary>
    /// <param name="groupName">The name of the group to delete.</param>
    /// <returns>Async task.</returns>
    public async Task TerminateAndDeleteAutoScalingGroupWithName(string groupName)
    {
        var describeGroupsResponse = await
_amazonAutoScaling.DescribeAutoScalingGroupsAsync(
            new DescribeAutoScalingGroupsRequest()
            {
                AutoScalingGroupNames = new List<string>() { groupName }
            });
        if (describeGroupsResponse.AutoScalingGroups.Any())
        {
            // Update the size to 0.
            await _amazonAutoScaling.UpdateAutoScalingGroupAsync(
                new UpdateAutoScalingGroupRequest()
                {
                    AutoScalingGroupName = groupName,
                    MinSize = 0
                });
            var group = describeGroupsResponse.AutoScalingGroups[0];
            foreach (var instance in group.Instances)
            {
                await TryTerminateInstanceById(instance.InstanceId);
            }

            await TryDeleteGroupByName(groupName);
        }
        else
        {
            Console.WriteLine($"No groups found with name {groupName}.");
        }
    }
```

```
    /// <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;
    }
```

- API の詳細については、「 API リファレンスDeleteAutoScalingGroup」の「」を参照してください。 AWS SDK for .NET

`DescribeAutoScalingGroups`

次の例は、`DescribeAutoScalingGroups` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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>();
        groups!.ForEach(group =>
        {
            if (group.AutoScalingGroupName == groupName)
            {
                group.Instances.ForEach(instance =>
                {
                    instanceIds.Add(instance.InstanceId);
                });
            }
        });

        var scalingGroupsRequest = new DescribeAutoScalingInstancesRequest
        {
            MaxRecords = 10,
            InstanceIds = instanceIds,
        };

        var response = await
 _amazonAutoScaling.DescribeAutoScalingInstancesAsync(scalingGroupsRequest);
        var instanceDetails = response.AutoScalingInstances;

        return instanceDetails;
    }
```

- API の詳細については、「 API リファレンスDescribeAutoScalingGroups」の「」を参照してください。 AWS SDK for .NET

DescribeAutoScalingInstances

次の例は、DescribeAutoScalingInstances を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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>();
        groups!.ForEach(group =>
        {
            if (group.AutoScalingGroupName == groupName)
            {
                group.Instances.ForEach(instance =>
                {
                    instanceIds.Add(instance.InstanceId);
                });
            }
        });

        var scalingGroupsRequest = new DescribeAutoScalingInstancesRequest
        {
            MaxRecords = 10,
            InstanceIds = instanceIds,
        };
```

```
        var response = await
 _amazonAutoScaling.DescribeAutoScalingInstancesAsync(scalingGroupsRequest);
        var instanceDetails = response.AutoScalingInstances;

        return instanceDetails;
    }
```

- API の詳細については、「 API リファレンスDescribeAutoScalingInstances」の「」を参照してください。 AWS SDK for .NET

`DescribeScalingActivities`

次の例は、`DescribeScalingActivities` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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<Amazon.AutoScaling.Model.Activity>>
 DescribeScalingActivitiesAsync(
        string groupName)
    {
        var scalingActivitiesRequest = new DescribeScalingActivitiesRequest
        {
            AutoScalingGroupName = groupName,
            MaxRecords = 10,
        };
```

```
        var response = await
_amazonAutoScaling.DescribeScalingActivitiesAsync(scalingActivitiesRequest);
        return response.Activities;
    }
```

- API の詳細については、「 API リファレンスDescribeScalingActivities」の「」を参照してください。 AWS SDK for .NET

`DisableMetricsCollection`

次の例は、DisableMetricsCollection を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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、「 API リファレンスDisableMetricsCollection」の「」を参照してください。 AWS SDK for .NET

`EnableMetricsCollection`

次の例は、EnableMetricsCollection を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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、「 API リファレンスEnableMetricsCollection」の「」を参照してください。 AWS SDK for .NET

`SetDesiredCapacity`

次の例は、`SetDesiredCapacity` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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、「 API リファレンスSetDesiredCapacity」の「」を参照してください。 AWS SDK for .NET

`TerminateInstanceInAutoScalingGroup`

次の例は、`TerminateInstanceInAutoScalingGroup` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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、「 API リファレンスTerminateInstanceInAutoScalingGroup」の「」を参照してください。 AWS SDK for .NET

`UpdateAutoScalingGroup`

次の例は、UpdateAutoScalingGroup を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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、「 API リファレンスUpdateAutoScalingGroup」の「」を参照してください。 AWS SDK for .NET

シナリオ

レジリエントなサービスの構築と管理

次のコード例は、本、映画、曲のレコメンデーションを返す負荷分散型ウェブサービスの作成方法を示しています。この例は、障害に対するサービスの対応方法と、障害発生時の耐障害性を高めるためにサービスを再構築する方法を示しています。

- Amazon EC2 Auto Scaling グループを使用して、起動テンプレートに基づいて Amazon Elastic Compute Cloud (Amazon EC2) インスタンスを作成し、インスタンス数を所定の範囲内に維持します。
- Elastic Load Balancing で HTTP リクエストを処理して配信します。

- Auto Scaling グループ内のインスタンスの状態を監視し、正常なインスタンスにのみリクエストを転送します。
- 各 EC2 インスタンスで Python ウェブサーバーを実行して HTTP リクエストを処理します。ウェブサーバーはレコメンデーションとヘルスチェックを返します。
- Amazon DynamoDB テーブルを使用してレコメンデーションサービスをシミュレートできます。
- AWS Systems Manager パラメータを更新して、リクエストとヘルスチェックに対するウェブサーバーの応答を制御します。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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のシナリオを実行します。

```
    static async Task Main(string[] args)
    {
        _configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally, load local settings.
            .Build();


        // Set up dependency injection for the AWS services.
        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<IAmazonIdentityManagementService>()
                    .AddAWSService<IAmazonDynamoDB>()
                    .AddAWSService<IAmazonElasticLoadBalancingV2>()
                    .AddAWSService<IAmazonSimpleSystemsManagement>()
```

```
                    .AddAWSService<IAmazonAutoScaling>()
                    .AddAWSService<IAmazonEC2>()
                    .AddTransient<AutoScalerWrapper>()
                    .AddTransient<ElasticLoadBalancerWrapper>()
                    .AddTransient<SmParameterWrapper>()
                    .AddTransient<Recommendations>()
                    .AddSingleton<IConfiguration>(_configuration)
            )
            .Build();

        ServicesSetup(host);
        ResourcesSetup();

        try
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Welcome to the Resilient Architecture Example
Scenario.");
            Console.WriteLine(new string('-', 80));
            await Deploy(true);

            Console.WriteLine("Now let's begin the scenario.");
            Console.WriteLine(new string('-', 80));
            await Demo(true);

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Finally, let's clean up our resources.");
            Console.WriteLine(new string('-', 80));

            await DestroyResources(true);

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Resilient Architecture Example Scenario is
complete.");
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
            await DestroyResources(true);
            Console.WriteLine(new string('-', 80));
        }
```

```
    }

    /// <summary>
    /// Setup any common resources, also used for integration testing.
    /// </summary>
    public static void ResourcesSetup()
    {
        _httpClient = new HttpClient();
    }

    /// <summary>
    /// Populate the services for use within the console application.
    /// </summary>
    /// <param name="host">The services host.</param>
    private static void ServicesSetup(IHost host)
    {
        _elasticLoadBalancerWrapper =
host.Services.GetRequiredService<ElasticLoadBalancerWrapper>();
        _iamClient =
host.Services.GetRequiredService<IAmazonIdentityManagementService>();
        _recommendations = host.Services.GetRequiredService<Recommendations>();
        _autoScalerWrapper = host.Services.GetRequiredService<AutoScalerWrapper>();
        _smParameterWrapper =
host.Services.GetRequiredService<SmParameterWrapper>();
    }

    /// <summary>
    /// Deploy necessary resources for the scenario.
    /// </summary>
    /// <param name="interactive">True to run as interactive.</param>
    /// <returns>True if successful.</returns>
    public static async Task<bool> Deploy(bool interactive)
    {
        var protocol = "HTTP";
        var port = 80;
        var sshPort = 22;

        Console.WriteLine(
            "\nFor this demo, we'll use the AWS SDK for .NET to create several AWS
resources\n" +
            "to set up a load-balanced web service endpoint and explore some ways to
make it resilient\n" +
            "against various kinds of failures.\n\n" +
            "Some of the resources create by this demo are:\n");
```

```
        Console.WriteLine(
            "\t* A DynamoDB table that the web service depends on to provide book,
movie, and song recommendations.");
        Console.WriteLine(
            "\t* An EC2 launch template that defines EC2 instances that each contain
a Python web server.");
        Console.WriteLine(
            "\t* An EC2 Auto Scaling group that manages EC2 instances across several
Availability Zones.");
        Console.WriteLine(
            "\t* An Elastic Load Balancing (ELB) load balancer that targets the Auto
Scaling group to distribute requests.");
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to start deploying
resources.");
        if (interactive)
            Console.ReadLine();

        // Create and populate the DynamoDB table.
        var databaseTableName = _configuration["databaseName"];
        var recommendationsPath = Path.Join(_configuration["resourcePath"],
            "recommendations_objects.json");
        Console.WriteLine($"Creating and populating a DynamoDB table named
{databaseTableName}.");
        await _recommendations.CreateDatabaseWithName(databaseTableName);
        await _recommendations.PopulateDatabase(databaseTableName,
recommendationsPath);
        Console.WriteLine(new string('-', 80));

        // Create the EC2 Launch Template.

        Console.WriteLine(
            $"Creating an EC2 launch template that runs 'server_startup_script.sh'
when an instance starts.\n"
            + "\nThis script starts a Python web server defined in the `server.py`
script. The web server\n"
            + "listens to HTTP requests on port 80 and responds to requests to '/'
and to '/healthcheck'.\n"
            + "For demo purposes, this server is run as the root user. In
production, the best practice is to\n"
            + "run a web server, such as Apache, with least-privileged
credentials.");
        Console.WriteLine(
```

```
            "\nThe template also defines an IAM policy that each instance uses to
assume a role that grants\n"
            + "permissions to access the DynamoDB recommendation table and Systems
Manager parameters\n"
            + "that control the flow of the demo.");

        var startupScriptPath = Path.Join(_configuration["resourcePath"],
            "server_startup_script.sh");
        var instancePolicyPath = Path.Join(_configuration["resourcePath"],
            "instance_policy.json");
        await _autoScalerWrapper.CreateTemplate(startupScriptPath,
instancePolicyPath);
        Console.WriteLine(new string('-', 80));

        Console.WriteLine(
            "Creating an EC2 Auto Scaling group that maintains three EC2 instances,
each in a different\n"
            + "Availability Zone.\n");
        var zones = await _autoScalerWrapper.DescribeAvailabilityZones();
        await _autoScalerWrapper.CreateGroupOfSize(3, _autoScalerWrapper.GroupName,
zones);
        Console.WriteLine(new string('-', 80));

        Console.WriteLine(
            "At this point, you have EC2 instances created. Once each instance
starts, it listens for\n"
            + "HTTP requests. You can see these instances in the console or continue
with the demo.\n");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to continue.");
        if (interactive)
            Console.ReadLine();

        Console.WriteLine("Creating variables that control the flow of the demo.");
        await _smParameterWrapper.Reset();

        Console.WriteLine(
            "\nCreating an Elastic Load Balancing target group and load balancer.
The target group\n"
            + "defines how the load balancer connects to instances. The load
balancer provides a\n"
            + "single endpoint where clients connect and dispatches requests to
instances in the group.");
```

```
        var defaultVpc = await _autoScalerWrapper.GetDefaultVpc();
        var subnets = await
 _autoScalerWrapper.GetAllVpcSubnetsForZones(defaultVpc.VpcId, zones);
        var subnetIds = subnets.Select(s => s.SubnetId).ToList();
        var targetGroup = await
 _elasticLoadBalancerWrapper.CreateTargetGroupOnVpc(_elasticLoadBalancerWrapper.TargetGroupN
 protocol, port, defaultVpc.VpcId);

        await
 _elasticLoadBalancerWrapper.CreateLoadBalancerAndListener(_elasticLoadBalancerWrapper.LoadB
 subnetIds, targetGroup);
        await
 _autoScalerWrapper.AttachLoadBalancerToGroup(_autoScalerWrapper.GroupName,
 targetGroup.TargetGroupArn);
        Console.WriteLine("\nVerifying access to the load balancer endpoint...");
        var endPoint = await
 _elasticLoadBalancerWrapper.GetEndpointForLoadBalancerByName(_elasticLoadBalancerWrapper.Lo
        var loadBalancerAccess = await
 _elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);

        if (!loadBalancerAccess)
        {
            Console.WriteLine("\nCouldn't connect to the load balancer, verifying
 that the port is open...");

            var ipString = await _httpClient.GetStringAsync("https://
checkip.amazonaws.com");
            ipString = ipString.Trim();

            var defaultSecurityGroup = await
 _autoScalerWrapper.GetDefaultSecurityGroupForVpc(defaultVpc);
            var portIsOpen =
 _autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, port, ipString);
            var sshPortIsOpen =
 _autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, sshPort,
 ipString);

            if (!portIsOpen)
            {
                Console.WriteLine(
                    "\nFor this example to work, the default security group for your
 default VPC must\n"
```

```
                    + "allows access from this computer. You can either add it
 automatically from this\n"
                    + "example or add it yourself using the AWS Management Console.
\n");

                if (!interactive || GetYesNoResponse(
                        "Do you want to add a rule to the security group to allow
 inbound traffic from your computer's IP address?"))
                {
                    await
 _autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, port, ipString);
                }
            }

            if (!sshPortIsOpen)
            {
                if (!interactive || GetYesNoResponse(
                        "Do you want to add a rule to the security group to allow
 inbound SSH traffic for debugging from your computer's IP address?"))
                {
                    await
 _autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, sshPort,
 ipString);
                }
            }
            loadBalancerAccess = await
 _elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);
        }

        if (loadBalancerAccess)
        {
            Console.WriteLine("Your load balancer is ready. You can access it by
 browsing to:");
            Console.WriteLine($"\thttp://{endPoint}\n");
        }
        else
        {
            Console.WriteLine(
                "\nCouldn't get a successful response from the load balancer
 endpoint. Troubleshoot by\n"
                + "manually verifying that your VPC and security group are
 configured correctly and that\n"
                + "you can successfully make a GET request to the load balancer
 endpoint:\n");
```

```
            Console.WriteLine($"\thttp://{endPoint}\n");
        }
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to continue with the
demo.");
        if (interactive)
            Console.ReadLine();
        return true;
    }

    /// <summary>
    /// Demonstrate the steps of the scenario.
    /// </summary>
    /// <param name="interactive">True to run as an interactive scenario.</param>
    /// <returns>Async task.</returns>
    public static async Task<bool> Demo(bool interactive)
    {
        var ssmOnlyPolicy = Path.Join(_configuration["resourcePath"],
            "ssm_only_policy.json");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Resetting parameters to starting values for demo.");
        await _smParameterWrapper.Reset();

        Console.WriteLine("\nThis part of the demonstration shows how to toggle
different parts of the system\n" +
                          "to create situations where the web service fails, and
shows how using a resilient\n" +
                          "architecture can keep the web service running in spite of
these failures.");
        Console.WriteLine(new string('-', 88));
        Console.WriteLine("At the start, the load balancer endpoint returns
recommendations and reports that all targets are healthy.");
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine($"The web service running on the EC2 instances gets
recommendations by querying a DynamoDB table.\n" +
                          $"The table name is contained in a Systems Manager
parameter named '{_smParameterWrapper.TableParameter}'.\n" +
                          $"To simulate a failure of the recommendation service,
let's set this parameter to name a non-existent table.\n");
```

```
        await
 _smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter, "this-
is-not-a-table");
        Console.WriteLine("\nNow, sending a GET request to the load balancer
 endpoint returns a failure code. But, the service reports as\n" +
                          "healthy to the load balancer because shallow health
 checks don't check for failure of the recommendation service.");
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("Instead of failing when the recommendation service fails,
 the web service can return a static response.");
        Console.WriteLine("While this is not a perfect solution, it presents the
 customer with a somewhat better experience than failure.");

        await
 _smParameterWrapper.PutParameterByName(_smParameterWrapper.FailureResponseParameter,
 "static");

        Console.WriteLine("\nNow, sending a GET request to the load balancer
 endpoint returns a static response.");
        Console.WriteLine("The service still reports as healthy because health
 checks are still shallow.");
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("Let's reinstate the recommendation service.\n");
        await
 _smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter,
 _smParameterWrapper.TableName);
        Console.WriteLine(
            "\nLet's also substitute bad credentials for one of the instances in the
 target group so that it can't\n" +
            "access the DynamoDB recommendation table.\n"
        );
        await _autoScalerWrapper.CreateInstanceProfileWithName(
            _autoScalerWrapper.BadCredsPolicyName,
            _autoScalerWrapper.BadCredsRoleName,
            _autoScalerWrapper.BadCredsProfileName,
            ssmOnlyPolicy,
            new List<string> { "AmazonSSMManagedInstanceCore" }
        );
        var instances = await
 _autoScalerWrapper.GetInstancesByGroupName(_autoScalerWrapper.GroupName);
```

```
        var badInstanceId = instances.First();
        var instanceProfile = await
_autoScalerWrapper.GetInstanceProfile(badInstanceId);
        Console.WriteLine(
            $"Replacing the profile for instance {badInstanceId} with a profile that
contains\n" +
            "bad credentials...\n"
        );
        await _autoScalerWrapper.ReplaceInstanceProfile(
            badInstanceId,
            _autoScalerWrapper.BadCredsProfileName,
            instanceProfile.AssociationId
        );
        Console.WriteLine(
            "Now, sending a GET request to the load balancer endpoint returns either
a recommendation or a static response,\n" +
            "depending on which instance is selected by the load balancer.\n"
        );
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nLet's implement a deep health check. For this demo, a
deep health check tests whether");
        Console.WriteLine("the web service can access the DynamoDB table that it
depends on for recommendations. Note that");
        Console.WriteLine("the deep health check is only for ELB routing and not for
Auto Scaling instance health.");
        Console.WriteLine("This kind of deep health check is not recommended for
Auto Scaling instance health, because it");
        Console.WriteLine("risks accidental termination of all instances in the Auto
Scaling group when a dependent service fails.");

        Console.WriteLine("\nBy implementing deep health checks, the load balancer
can detect when one of the instances is failing");
        Console.WriteLine("and take that instance out of rotation.");

        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.HealthCheckParameter,
"deep");

        Console.WriteLine($"\nNow, checking target health indicates that the
instance with bad credentials ({badInstanceId})");
        Console.WriteLine("is unhealthy. Note that it might take a minute or two for
the load balancer to detect the unhealthy");
```

```
        Console.WriteLine("instance. Sending a GET request to the load balancer
 endpoint always returns a recommendation, because");
        Console.WriteLine("the load balancer takes unhealthy instances out of its
 rotation.");

        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nBecause the instances in this demo are controlled by an
 auto scaler, the simplest way to fix an unhealthy");
        Console.WriteLine("instance is to terminate it and let the auto scaler start
 a new instance to replace it.");

        await _autoScalerWrapper.TryTerminateInstanceById(badInstanceId);

        Console.WriteLine($"\nEven while the instance is terminating and the new
 instance is starting, sending a GET");
        Console.WriteLine("request to the web service continues to get a successful
 recommendation response because");
        Console.WriteLine("starts and reports as healthy, it is included in the load
 balancing rotation.");
        Console.WriteLine("Note that terminating and replacing an instance typically
 takes several minutes, during which time you");
        Console.WriteLine("can see the changing health check status until the new
 instance is running and healthy.");

        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nIf the recommendation service fails now, deep health
 checks mean all instances report as unhealthy.");

        await
 _smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter, "this-
is-not-a-table");

        Console.WriteLine($"\nWhen all instances are unhealthy, the load balancer
 continues to route requests even to");
        Console.WriteLine("unhealthy instances, allowing them to fail open and
 return a static response rather than fail");
        Console.WriteLine("closed and report failure to the customer.");

        if (interactive)
            await DemoActionChoices();
```

```
        await _smParameterWrapper.Reset();

        Console.WriteLine(new string('-', 80));
        return true;
    }

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <param name="interactive">True to ask the user for cleanup.</param>
    /// <returns>Async task.</returns>
    public static async Task<bool> DestroyResources(bool interactive)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine(
            "To keep things tidy and to avoid unwanted charges on your account, we
can clean up all AWS resources\n" +
            "that were created for this demo."
        );

        if (!interactive || GetYesNoResponse("Do you want to clean up all demo
resources? (y/n) "))
        {
            await
_elasticLoadBalancerWrapper.DeleteLoadBalancerByName(_elasticLoadBalancerWrapper.LoadBalanc
            await
_elasticLoadBalancerWrapper.DeleteTargetGroupByName(_elasticLoadBalancerWrapper.TargetGroup
            await
_autoScalerWrapper.TerminateAndDeleteAutoScalingGroupWithName(_autoScalerWrapper.GroupName)
            await
_autoScalerWrapper.DeleteKeyPairByName(_autoScalerWrapper.KeyPairName);
            await
_autoScalerWrapper.DeleteTemplateByName(_autoScalerWrapper.LaunchTemplateName);
            await _autoScalerWrapper.DeleteInstanceProfile(
                _autoScalerWrapper.BadCredsProfileName,
                _autoScalerWrapper.BadCredsRoleName
            );
            await
_recommendations.DestroyDatabaseByName(_recommendations.TableName);
        }
        else
        {
            Console.WriteLine(
                "Ok, we'll leave the resources intact.\n" +
```

```
                "Don't forget to delete them when you're done with them or you might
 incur unexpected charges."
            );
        }

        Console.WriteLine(new string('-', 80));
        return true;
    }
```

Auto Scaling と Amazon EC2 のアクションをラップするクラスを作成します。

```
/// <summary>
/// Encapsulates Amazon EC2 Auto Scaling and EC2 management methods.
/// </summary>
public class AutoScalerWrapper
{
    private readonly IAmazonAutoScaling _amazonAutoScaling;
    private readonly IAmazonEC2 _amazonEc2;
    private readonly IAmazonSimpleSystemsManagement _amazonSsm;
    private readonly IAmazonIdentityManagementService _amazonIam;

    private readonly string _instanceType = "";
    private readonly string _amiParam = "";
    private readonly string _launchTemplateName = "";
    private readonly string _groupName = "";
    private readonly string _instancePolicyName = "";
    private readonly string _instanceRoleName = "";
    private readonly string _instanceProfileName = "";
    private readonly string _badCredsProfileName = "";
    private readonly string _badCredsRoleName = "";
    private readonly string _badCredsPolicyName = "";
    private readonly string _keyPairName = "";

    public string GroupName => _groupName;
    public string KeyPairName => _keyPairName;
    public string LaunchTemplateName => _launchTemplateName;
    public string InstancePolicyName => _instancePolicyName;
    public string BadCredsProfileName => _badCredsProfileName;
    public string BadCredsRoleName => _badCredsRoleName;
    public string BadCredsPolicyName => _badCredsPolicyName;

    /// <summary>
```

```
    /// Constructor for the AutoScalerWrapper.
    /// </summary>
    /// <param name="amazonAutoScaling">The injected AutoScaling client.</param>
    /// <param name="amazonEc2">The injected EC2 client.</param>
    /// <param name="amazonIam">The injected IAM client.</param>
    /// <param name="amazonSsm">The injected SSM client.</param>
    public AutoScalerWrapper(
        IAmazonAutoScaling amazonAutoScaling,
        IAmazonEC2 amazonEc2,
        IAmazonSimpleSystemsManagement amazonSsm,
        IAmazonIdentityManagementService amazonIam,
        IConfiguration configuration)
    {
        _amazonAutoScaling = amazonAutoScaling;
        _amazonEc2 = amazonEc2;
        _amazonSsm = amazonSsm;
        _amazonIam = amazonIam;

        var prefix = configuration["resourcePrefix"];
        _instanceType = configuration["instanceType"];
        _amiParam = configuration["amiParam"];

        _launchTemplateName = prefix + "-template";
        _groupName = prefix + "-group";
        _instancePolicyName = prefix + "-pol";
        _instanceRoleName = prefix + "-role";
        _instanceProfileName = prefix + "-prof";
        _badCredsPolicyName = prefix + "-bc-pol";
        _badCredsRoleName = prefix + "-bc-role";
        _badCredsProfileName = prefix + "-bc-prof";
        _keyPairName = prefix + "-key-pair";
    }

    /// <summary>
    /// Create a policy, role, and profile that is associated with instances with a
specified name.
    /// An instance's associated profile defines a role that is assumed by the
    /// instance.The role has attached policies that specify the AWS permissions
granted to
    /// clients that run on the instance.
    /// </summary>
    /// <param name="policyName">Name to use for the policy.</param>
    /// <param name="roleName">Name to use for the role.</param>
    /// <param name="profileName">Name to use for the profile.</param>
```

```
    /// <param name="ssmOnlyPolicyFile">Path to a policy file for SSM.</param>
    /// <param name="awsManagedPolicies">AWS Managed policies to be attached to the
role.</param>
    /// <returns>The Arn of the profile.</returns>
    public async Task<string> CreateInstanceProfileWithName(
        string policyName,
        string roleName,
        string profileName,
        string ssmOnlyPolicyFile,
        List<string>? awsManagedPolicies = null)
    {

        var assumeRoleDoc = "{" +
                                "\"Version\": \"2012-10-17\"," +
                                "\"Statement\": [{" +
                                     "\"Effect\": \"Allow\"," +
                                     "\"Principal\": {" +
                                     "\"Service\": [" +
                                         "\"ec2.amazonaws.com\"" +
                                     "]" +
                                     "}," +
                                "\"Action\": \"sts:AssumeRole\"" +
                                "}]" +
                            "}";

        var policyDocument = await File.ReadAllTextAsync(ssmOnlyPolicyFile);

        var policyArn = "";

        try
        {
            var createPolicyResult = await _amazonIam.CreatePolicyAsync(
                new CreatePolicyRequest
                {
                    PolicyName = policyName,
                    PolicyDocument = policyDocument
                });
            policyArn = createPolicyResult.Policy.Arn;
        }
        catch (EntityAlreadyExistsException)
        {
            // The policy already exists, so we look it up to get the Arn.
            var policiesPaginator = _amazonIam.Paginators.ListPolicies(
                new ListPoliciesRequest()
```

```
                {
                    Scope = PolicyScopeType.Local
                });
            // Get the entire list using the paginator.
            await foreach (var policy in policiesPaginator.Policies)
            {
                if (policy.PolicyName.Equals(policyName))
                {
                    policyArn = policy.Arn;
                }
            }

            if (policyArn == null)
            {
                throw new InvalidOperationException("Policy not found");
            }
        }

        try
        {
            await _amazonIam.CreateRoleAsync(new CreateRoleRequest()
            {
                RoleName = roleName,
                AssumeRolePolicyDocument = assumeRoleDoc,
            });
            await _amazonIam.AttachRolePolicyAsync(new AttachRolePolicyRequest()
            {
                RoleName = roleName,
                PolicyArn = policyArn
            });
            if (awsManagedPolicies != null)
            {
                foreach (var awsPolicy in awsManagedPolicies)
                {
                    await _amazonIam.AttachRolePolicyAsync(new
AttachRolePolicyRequest()
                    {
                        PolicyArn = $"arn:aws:iam::aws:policy/{awsPolicy}",
                        RoleName = roleName
                    });
                }
            }
        }
        catch (EntityAlreadyExistsException)
```

```
        {
            Console.WriteLine("Role already exists.");
        }

        string profileArn = "";
        try
        {
            var profileCreateResponse = await _amazonIam.CreateInstanceProfileAsync(
                new CreateInstanceProfileRequest()
                {
                    InstanceProfileName = profileName
                });
            // Allow time for the profile to be ready.
            profileArn = profileCreateResponse.InstanceProfile.Arn;
            Thread.Sleep(10000);
            await _amazonIam.AddRoleToInstanceProfileAsync(
                new AddRoleToInstanceProfileRequest()
                {
                    InstanceProfileName = profileName,
                    RoleName = roleName
                });

        }
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine("Policy already exists.");
            var profileGetResponse = await _amazonIam.GetInstanceProfileAsync(
                new GetInstanceProfileRequest()
                {
                    InstanceProfileName = profileName
                });
            profileArn = profileGetResponse.InstanceProfile.Arn;
        }
        return profileArn;
    }

    /// <summary>
    /// Create a new key pair and save the file.
    /// </summary>
    /// <param name="newKeyPairName">The name of the new key pair.</param>
    /// <returns>Async task.</returns>
    public async Task CreateKeyPair(string newKeyPairName)
    {
        try
```

```
        {
            var keyResponse = await _amazonEc2.CreateKeyPairAsync(
                new CreateKeyPairRequest() { KeyName = newKeyPairName });
            await File.WriteAllTextAsync($"{newKeyPairName}.pem",
                keyResponse.KeyPair.KeyMaterial);
            Console.WriteLine($"Created key pair {newKeyPairName}.");
        }
        catch (AlreadyExistsException)
        {
            Console.WriteLine("Key pair already exists.");
        }
    }

    /// <summary>
    /// Delete the key pair and file by name.
    /// </summary>
    /// <param name="deleteKeyPairName">The key pair to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteKeyPairByName(string deleteKeyPairName)
    {
        try
        {
            await _amazonEc2.DeleteKeyPairAsync(
                new DeleteKeyPairRequest() { KeyName = deleteKeyPairName });
            File.Delete($"{deleteKeyPairName}.pem");
        }
        catch (FileNotFoundException)
        {
            Console.WriteLine($"Key pair {deleteKeyPairName} not found.");
        }
    }

    /// <summary>
    /// Creates an Amazon EC2 launch template to use with Amazon EC2 Auto Scaling.
    /// The launch template specifies a Bash script in its user data field that runs
after
    /// the instance is started. This script installs the Python packages and starts
a Python
    /// web server on the instance.
    /// </summary>
    /// <param name="startupScriptPath">The path to a Bash script file that is
run.</param>
    /// <param name="instancePolicyPath">The path to a permissions policy to create
and attach to the profile.</param>
```

```
    /// <returns>The template object.</returns>
    public async Task<Amazon.EC2.Model.LaunchTemplate> CreateTemplate(string
startupScriptPath, string instancePolicyPath)
    {
        await CreateKeyPair(_keyPairName);
        await CreateInstanceProfileWithName(_instancePolicyName, _instanceRoleName,
_instanceProfileName, instancePolicyPath);

        var startServerText = await File.ReadAllTextAsync(startupScriptPath);
        var plainTextBytes = System.Text.Encoding.UTF8.GetBytes(startServerText);

        var amiLatest = await _amazonSsm.GetParameterAsync(
            new GetParameterRequest() { Name = _amiParam });
        var amiId = amiLatest.Parameter.Value;
        var launchTemplateResponse = await _amazonEc2.CreateLaunchTemplateAsync(
            new CreateLaunchTemplateRequest()
            {
                LaunchTemplateName = _launchTemplateName,
                LaunchTemplateData = new RequestLaunchTemplateData()
                {
                    InstanceType = _instanceType,
                    ImageId = amiId,
                    IamInstanceProfile =
                        new
                            LaunchTemplateIamInstanceProfileSpecificationRequest()
                        {
                            Name = _instanceProfileName
                        },
                    KeyName = _keyPairName,
                    UserData = System.Convert.ToBase64String(plainTextBytes)
                }
            });
        return launchTemplateResponse.LaunchTemplate;

    }

    /// <summary>
    /// Get a list of Availability Zones in the AWS Region of the Amazon EC2 Client.
    /// </summary>
    /// <returns>A list of availability zones.</returns>
    public async Task<List<string>> DescribeAvailabilityZones()
    {
        var zoneResponse = await _amazonEc2.DescribeAvailabilityZonesAsync(
            new DescribeAvailabilityZonesRequest());
```

```
        return zoneResponse.AvailabilityZones.Select(z => z.ZoneName).ToList();
    }

    /// <summary>
    /// Create an EC2 Auto Scaling group of a specified size and name.
    /// </summary>
    /// <param name="groupSize">The size for the group.</param>
    /// <param name="groupName">The name for the group.</param>
    /// <param name="availabilityZones">The availability zones for the group.</
param>
    /// <returns>Async task.</returns>
    public async Task CreateGroupOfSize(int groupSize, string groupName,
 List<string> availabilityZones)
    {
        try
        {
            await _amazonAutoScaling.CreateAutoScalingGroupAsync(
                new CreateAutoScalingGroupRequest()
                {
                    AutoScalingGroupName = groupName,
                    AvailabilityZones = availabilityZones,
                    LaunchTemplate =
                        new Amazon.AutoScaling.Model.LaunchTemplateSpecification()
                        {
                            LaunchTemplateName = _launchTemplateName,
                            Version = "$Default"
                        },
                    MaxSize = groupSize,
                    MinSize = groupSize
                });
            Console.WriteLine($"Created EC2 Auto Scaling group {groupName} with size
 {groupSize}.");
        }
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine($"EC2 Auto Scaling group {groupName} already
 exists.");
        }
    }

    /// <summary>
    /// Get the default VPC for the account.
    /// </summary>
    /// <returns>The default VPC object.</returns>
```

```
    public async Task<Vpc> GetDefaultVpc()
    {
        var vpcResponse = await _amazonEc2.DescribeVpcsAsync(
            new DescribeVpcsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("is-default", new List<string>() { "true" })
                }
            });
        return vpcResponse.Vpcs[0];
    }

    /// <summary>
    /// Get all the subnets for a Vpc in a set of availability zones.
    /// </summary>
    /// <param name="vpcId">The Id of the Vpc.</param>
    /// <param name="availabilityZones">The list of availability zones.</param>
    /// <returns>The collection of subnet objects.</returns>
    public async Task<List<Subnet>> GetAllVpcSubnetsForZones(string vpcId,
List<string> availabilityZones)
    {
        var subnets = new List<Subnet>();
        var subnetPaginator = _amazonEc2.Paginators.DescribeSubnets(
            new DescribeSubnetsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("vpc-id", new List<string>() { vpcId}),
                    new ("availability-zone", availabilityZones),
                    new ("default-for-az", new List<string>() { "true" })
                }
            });

        // Get the entire list using the paginator.
        await foreach (var subnet in subnetPaginator.Subnets)
        {
            subnets.Add(subnet);
        }

        return subnets;
    }

    /// <summary>
```

```
    /// Delete a launch template by name.
    /// </summary>
    /// <param name="templateName">The name of the template to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteTemplateByName(string templateName)
    {
        try
        {
            await _amazonEc2.DeleteLaunchTemplateAsync(
                new DeleteLaunchTemplateRequest()
                {
                    LaunchTemplateName = templateName
                });
        }
        catch (AmazonClientException)
        {
            Console.WriteLine($"Unable to delete template {templateName}.");
        }
    }

    /// <summary>
    /// Detaches a role from an instance profile, detaches policies from the role,
    /// and deletes all the resources.
    /// </summary>
    /// <param name="profileName">The name of the profile to delete.</param>
    /// <param name="roleName">The name of the role to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteInstanceProfile(string profileName, string roleName)
    {
        try
        {
            await _amazonIam.RemoveRoleFromInstanceProfileAsync(
                new RemoveRoleFromInstanceProfileRequest()
                {
                    InstanceProfileName = profileName,
                    RoleName = roleName
                });
            await _amazonIam.DeleteInstanceProfileAsync(
                new DeleteInstanceProfileRequest() { InstanceProfileName =
profileName });
            var attachedPolicies = await _amazonIam.ListAttachedRolePoliciesAsync(
                new ListAttachedRolePoliciesRequest() { RoleName = roleName });
            foreach (var policy in attachedPolicies.AttachedPolicies)
            {
```

```
                await _amazonIam.DetachRolePolicyAsync(
                    new DetachRolePolicyRequest()
                    {
                        RoleName = roleName,
                        PolicyArn = policy.PolicyArn
                    });
                // Delete the custom policies only.
                if (!policy.PolicyArn.StartsWith("arn:aws:iam::aws"))
                {
                    await _amazonIam.DeletePolicyAsync(
                        new Amazon.IdentityManagement.Model.DeletePolicyRequest()
                        {
                            PolicyArn = policy.PolicyArn
                        });
                }
            }

            await _amazonIam.DeleteRoleAsync(
                new DeleteRoleRequest() { RoleName = roleName });
        }
        catch (NoSuchEntityException)
        {
            Console.WriteLine($"Instance profile {profileName} does not exist.");
        }
    }

    /// <summary>
    /// Gets data about the instances in an EC2 Auto Scaling group by its group
name.
    /// </summary>
    /// <param name="group">The name of the auto scaling group.</param>
    /// <returns>A collection of instance Ids.</returns>
    public async Task<IEnumerable<string>> GetInstancesByGroupName(string group)
    {
        var instanceResponse = await
_amazonAutoScaling.DescribeAutoScalingGroupsAsync(
            new DescribeAutoScalingGroupsRequest()
            {
                AutoScalingGroupNames = new List<string>() { group }
            });
        var instanceIds = instanceResponse.AutoScalingGroups.SelectMany(
            g => g.Instances.Select(i => i.InstanceId));
        return instanceIds;
    }
```

```
    /// <summary>
    /// Get the instance profile association data for an instance.
    /// </summary>
    /// <param name="instanceId">The Id of the instance.</param>
    /// <returns>Instance profile associations data.</returns>
    public async Task<IamInstanceProfileAssociation> GetInstanceProfile(string
instanceId)
    {
        var response = await _amazonEc2.DescribeIamInstanceProfileAssociationsAsync(
            new DescribeIamInstanceProfileAssociationsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("instance-id", new List<string>() { instanceId })
                },
            });
        return response.IamInstanceProfileAssociations[0];
    }

    /// <summary>
    /// Replace the profile associated with a running instance. After the profile is
replaced, the instance
    /// is rebooted to ensure that it uses the new profile. When the instance is
ready, Systems Manager is
    /// used to restart the Python web server.
    /// </summary>
    /// <param name="instanceId">The Id of the instance to update.</param>
    /// <param name="credsProfileName">The name of the new profile to associate with
the specified instance.</param>
    /// <param name="associationId">The Id of the existing profile association for
the instance.</param>
    /// <returns>Async task.</returns>
    public async Task ReplaceInstanceProfile(string instanceId, string
credsProfileName, string associationId)
    {
        await _amazonEc2.ReplaceIamInstanceProfileAssociationAsync(
                new ReplaceIamInstanceProfileAssociationRequest()
                {
                    AssociationId = associationId,
                    IamInstanceProfile = new IamInstanceProfileSpecification()
                    {
                        Name = credsProfileName
                    }
```

```
                });
        // Allow time before resetting.
        Thread.Sleep(25000);
        var instanceReady = false;
        var retries = 5;
        while (retries-- > 0 && !instanceReady)
        {
            await _amazonEc2.RebootInstancesAsync(
                new RebootInstancesRequest(new List<string>() { instanceId }));
            Thread.Sleep(10000);

            var instancesPaginator =
_amazonSsm.Paginators.DescribeInstanceInformation(
                new DescribeInstanceInformationRequest());
            // Get the entire list using the paginator.
            await foreach (var instance in
instancesPaginator.InstanceInformationList)
            {
                instanceReady = instance.InstanceId == instanceId;
                if (instanceReady)
                {
                    break;
                }
            }
        }
        Console.WriteLine($"Sending restart command to instance {instanceId}");
        await _amazonSsm.SendCommandAsync(
            new SendCommandRequest()
            {
                InstanceIds = new List<string>() { instanceId },
                DocumentName = "AWS-RunShellScript",
                Parameters = new Dictionary<string, List<string>>()
                {
                        {"commands", new List<string>() { "cd / && sudo python3
server.py 80" }}
                }
            });
        Console.WriteLine($"Restarted the web server on instance {instanceId}");
    }

    /// <summary>
    /// Try to terminate an instance by its Id.
    /// </summary>
    /// <param name="instanceId">The Id of the instance to terminate.</param>
```

```
    /// <returns>Async task.</returns>
    public async Task TryTerminateInstanceById(string instanceId)
    {
        var stopping = false;
        Console.WriteLine($"Stopping {instanceId}...");
        while (!stopping)
        {
            try
            {
                await _amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(
                    new TerminateInstanceInAutoScalingGroupRequest()
                    {
                        InstanceId = instanceId,
                        ShouldDecrementDesiredCapacity = false
                    });
                stopping = true;
            }
            catch (ScalingActivityInProgressException)
            {
                Console.WriteLine($"Scaling activity in progress for {instanceId}.
Waiting...");
                Thread.Sleep(10000);
            }
        }
    }

    /// <summary>
    /// Tries to delete the EC2 Auto Scaling group. If the group is in use or in
progress,
    /// waits and retries until the group is successfully deleted.
    /// </summary>
    /// <param name="groupName">The name of the group to try to delete.</param>
    /// <returns>Async task.</returns>
    public async Task TryDeleteGroupByName(string groupName)
    {
        var stopped = false;
        while (!stopped)
        {
            try
            {
                await _amazonAutoScaling.DeleteAutoScalingGroupAsync(
                    new DeleteAutoScalingGroupRequest()
                    {
                        AutoScalingGroupName = groupName
```

```
                    });
                stopped = true;
            }
            catch (Exception e)
                when ((e is ScalingActivityInProgressException)
                      || (e is Amazon.AutoScaling.Model.ResourceInUseException))
            {
                Console.WriteLine($"Some instances are still running. Waiting...");
                Thread.Sleep(10000);
            }
        }
    }

    /// <summary>
    /// Terminate instances and delete the Auto Scaling group by name.
    /// </summary>
    /// <param name="groupName">The name of the group to delete.</param>
    /// <returns>Async task.</returns>
    public async Task TerminateAndDeleteAutoScalingGroupWithName(string groupName)
    {
        var describeGroupsResponse = await
_amazonAutoScaling.DescribeAutoScalingGroupsAsync(
            new DescribeAutoScalingGroupsRequest()
            {
                AutoScalingGroupNames = new List<string>() { groupName }
            });
        if (describeGroupsResponse.AutoScalingGroups.Any())
        {
            // Update the size to 0.
            await _amazonAutoScaling.UpdateAutoScalingGroupAsync(
                new UpdateAutoScalingGroupRequest()
                {
                    AutoScalingGroupName = groupName,
                    MinSize = 0
                });
            var group = describeGroupsResponse.AutoScalingGroups[0];
            foreach (var instance in group.Instances)
            {
                await TryTerminateInstanceById(instance.InstanceId);
            }

            await TryDeleteGroupByName(groupName);
        }
        else
```

```
        {
            Console.WriteLine($"No groups found with name {groupName}.");
        }
    }


    /// <summary>
    /// Get the default security group for a specified Vpc.
    /// </summary>
    /// <param name="vpc">The Vpc to search.</param>
    /// <returns>The default security group.</returns>
    public async Task<SecurityGroup> GetDefaultSecurityGroupForVpc(Vpc vpc)
    {
        var groupResponse = await _amazonEc2.DescribeSecurityGroupsAsync(
            new DescribeSecurityGroupsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("group-name", new List<string>() { "default" }),
                    new ("vpc-id", new List<string>() { vpc.VpcId })
                }
            });
        return groupResponse.SecurityGroups[0];
    }

    /// <summary>
    /// Verify the default security group of a Vpc allows ingress from the calling
computer.
    /// This can be done by allowing ingress from this computer's IP address.
    /// In some situations, such as connecting from a corporate network, you must
instead specify
    /// a prefix list Id. You can also temporarily open the port to any IP address
while running this example.
    /// If you do, be sure to remove public access when you're done.
    /// </summary>
    /// <param name="vpc">The group to check.</param>
    /// <param name="port">The port to verify.</param>
    /// <param name="ipAddress">This computer's IP address.</param>
    /// <returns>True if the ip address is allowed on the group.</returns>
    public bool VerifyInboundPortForGroup(SecurityGroup group, int port, string
ipAddress)
    {
        var portIsOpen = false;
        foreach (var ipPermission in group.IpPermissions)
```

```
        {
            if (ipPermission.FromPort == port)
            {
                foreach (var ipRange in ipPermission.Ipv4Ranges)
                {
                    var cidr = ipRange.CidrIp;
                    if (cidr.StartsWith(ipAddress) || cidr == "0.0.0.0/0")
                    {
                        portIsOpen = true;
                    }
                }

                if (ipPermission.PrefixListIds.Any())
                {
                    portIsOpen = true;
                }

                if (!portIsOpen)
                {
                    Console.WriteLine("The inbound rule does not appear to be open
to either this computer's IP\n" +
                                      "address, to all IP addresses (0.0.0.0/0), or
to a prefix list ID.");
                }
                else
                {
                    break;
                }
            }
        }

        return portIsOpen;
    }

    /// <summary>
    /// Add an ingress rule to the specified security group that allows access on
the
    /// specified port from the specified IP address.
    /// </summary>
    /// <param name="groupId">The Id of the security group to modify.</param>
    /// <param name="port">The port to open.</param>
    /// <param name="ipAddress">The IP address to allow access.</param>
    /// <returns>Async task.</returns>
    public async Task OpenInboundPort(string groupId, int port, string ipAddress)
```

```
    {
        await _amazonEc2.AuthorizeSecurityGroupIngressAsync(
            new AuthorizeSecurityGroupIngressRequest()
            {
                GroupId = groupId,
                IpPermissions = new List<IpPermission>()
                {
                    new IpPermission()
                    {
                        FromPort = port,
                        ToPort = port,
                        IpProtocol = "tcp",
                        Ipv4Ranges = new List<IpRange>()
                        {
                            new IpRange() { CidrIp = $"{ipAddress}/32" }
                        }
                    }
                }
            });
    }

    /// <summary>
    /// Attaches an Elastic Load Balancing (ELB) target group to this EC2 Auto
 Scaling group.
    /// The
    /// </summary>
    /// <param name="autoScalingGroupName">The name of the Auto Scaling group.</
param>
    /// <param name="targetGroupArn">The Arn for the target group.</param>
    /// <returns>Async task.</returns>
    public async Task AttachLoadBalancerToGroup(string autoScalingGroupName, string
 targetGroupArn)
    {
        await _amazonAutoScaling.AttachLoadBalancerTargetGroupsAsync(
            new AttachLoadBalancerTargetGroupsRequest()
            {
                AutoScalingGroupName = autoScalingGroupName,
                TargetGroupARNs = new List<string>() { targetGroupArn }
            });
    }
}
```

Elastic Load Balancing のアクションをラップするクラスを作成します。

```
/// <summary>
/// Encapsulates Elastic Load Balancer actions.
/// </summary>
public class ElasticLoadBalancerWrapper
{
    private readonly IAmazonElasticLoadBalancingV2 _amazonElasticLoadBalancingV2;
    private string? _endpoint = null;
    private readonly string _targetGroupName = "";
    private readonly string _loadBalancerName = "";
    HttpClient _httpClient = new();

    public string TargetGroupName => _targetGroupName;
    public string LoadBalancerName => _loadBalancerName;

    /// <summary>
    /// Constructor for the Elastic Load Balancer wrapper.
    /// </summary>
    /// <param name="amazonElasticLoadBalancingV2">The injected load balancing v2
 client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public ElasticLoadBalancerWrapper(
        IAmazonElasticLoadBalancingV2 amazonElasticLoadBalancingV2,
        IConfiguration configuration)
    {
        _amazonElasticLoadBalancingV2 = amazonElasticLoadBalancingV2;
        var prefix = configuration["resourcePrefix"];
        _targetGroupName = prefix + "-tg";
        _loadBalancerName = prefix + "-lb";
    }

    /// <summary>
    /// Get the HTTP Endpoint of a load balancer by its name.
    /// </summary>
    /// <param name="loadBalancerName">The name of the load balancer.</param>
    /// <returns>The HTTP endpoint.</returns>
    public async Task<string> GetEndpointForLoadBalancerByName(string
 loadBalancerName)
    {
        if (_endpoint == null)
        {
            var endpointResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
```

```
                    new DescribeLoadBalancersRequest()
                    {
                        Names = new List<string>() { loadBalancerName }
                    });
            _endpoint = endpointResponse.LoadBalancers[0].DNSName;
        }

        return _endpoint;
    }

    /// <summary>
    /// Return the GET response for an endpoint as text.
    /// </summary>
    /// <param name="endpoint">The endpoint for the request.</param>
    /// <returns>The request response.</returns>
    public async Task<string> GetEndPointResponse(string endpoint)
    {
        var endpointResponse = await _httpClient.GetAsync($"http://{endpoint}");
        var textResponse = await endpointResponse.Content.ReadAsStringAsync();
        return textResponse!;
    }

    /// <summary>
    /// Get the target health for a group by name.
    /// </summary>
    /// <param name="groupName">The name of the group.</param>
    /// <returns>The collection of health descriptions.</returns>
    public async Task<List<TargetHealthDescription>>
CheckTargetHealthForGroup(string groupName)
    {
        List<TargetHealthDescription> result = null!;
        try
        {
            var groupResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                    new DescribeTargetGroupsRequest()
                    {
                        Names = new List<string>() { groupName }
                    });
            var healthResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetHealthAsync(
                    new DescribeTargetHealthRequest()
                    {
```

```
                        TargetGroupArn =
 groupResponse.TargetGroups[0].TargetGroupArn
                    });
            ;
            result = healthResponse.TargetHealthDescriptions;
        }
        catch (TargetGroupNotFoundException)
        {
            Console.WriteLine($"Target group {groupName} not found.");
        }
        return result;
    }

    /// <summary>
    /// Create an Elastic Load Balancing target group. The target group specifies
 how the load balancer forwards
    /// requests to instances in the group and how instance health is checked.
    ///
    /// To speed up this demo, the health check is configured with shortened times
 and lower thresholds. In production,
    /// you might want to decrease the sensitivity of your health checks to avoid
 unwanted failures.
    /// </summary>
    /// <param name="groupName">The name for the group.</param>
    /// <param name="protocol">The protocol, such as HTTP.</param>
    /// <param name="port">The port to use to forward requests, such as 80.</param>
    /// <param name="vpcId">The Id of the Vpc in which the load balancer exists.</
param>
    /// <returns>The new TargetGroup object.</returns>
    public async Task<TargetGroup> CreateTargetGroupOnVpc(string groupName,
 ProtocolEnum protocol, int port, string vpcId)
    {
        var createResponse = await
 _amazonElasticLoadBalancingV2.CreateTargetGroupAsync(
            new CreateTargetGroupRequest()
            {
                Name = groupName,
                Protocol = protocol,
                Port = port,
                HealthCheckPath = "/healthcheck",
                HealthCheckIntervalSeconds = 10,
                HealthCheckTimeoutSeconds = 5,
                HealthyThresholdCount = 2,
                UnhealthyThresholdCount = 2,
```

```
                VpcId = vpcId
            });
        var targetGroup = createResponse.TargetGroups[0];
        return targetGroup;
    }

    /// <summary>
    /// Create an Elastic Load Balancing load balancer that uses the specified
subnets
    /// and forwards requests to the specified target group.
    /// </summary>
    /// <param name="name">The name for the new load balancer.</param>
    /// <param name="subnetIds">Subnets for the load balancer.</param>
    /// <param name="targetGroup">Target group for forwarded requests.</param>
    /// <returns>The new LoadBalancer object.</returns>
    public async Task<LoadBalancer> CreateLoadBalancerAndListener(string name,
List<string> subnetIds, TargetGroup targetGroup)
    {
        var createLbResponse = await
_amazonElasticLoadBalancingV2.CreateLoadBalancerAsync(
            new CreateLoadBalancerRequest()
            {
                Name = name,
                Subnets = subnetIds
            });
        var loadBalancerArn = createLbResponse.LoadBalancers[0].LoadBalancerArn;

        // Wait for load balancer to be available.
        var loadBalancerReady = false;
        while (!loadBalancerReady)
        {
            try
            {
                var describeResponse =
                    await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                        new DescribeLoadBalancersRequest()
                        {
                            Names = new List<string>() { name }
                        });

                var loadBalancerState =
describeResponse.LoadBalancers[0].State.Code;
```

```
                loadBalancerReady = loadBalancerState ==
 LoadBalancerStateEnum.Active;
            }
            catch (LoadBalancerNotFoundException)
            {
                loadBalancerReady = false;
            }
            Thread.Sleep(10000);
        }
        // Create the listener.
        await _amazonElasticLoadBalancingV2.CreateListenerAsync(
            new CreateListenerRequest()
            {
                LoadBalancerArn = loadBalancerArn,
                Protocol = targetGroup.Protocol,
                Port = targetGroup.Port,
                DefaultActions = new List<Action>()
                {
                    new Action()
                    {
                        Type = ActionTypeEnum.Forward,
                        TargetGroupArn = targetGroup.TargetGroupArn
                    }
                }
            });
        return createLbResponse.LoadBalancers[0];
    }

    /// <summary>
    /// Verify this computer can successfully send a GET request to the
    /// load balancer endpoint.
    /// </summary>
    /// <param name="endpoint">The endpoint to check.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> VerifyLoadBalancerEndpoint(string endpoint)
    {
        var success = false;
        var retries = 3;
        while (!success && retries > 0)
        {
            try
            {
                var endpointResponse = await _httpClient.GetAsync($"http://
{endpoint}");
```

```
                Console.WriteLine($"Response: {endpointResponse.StatusCode}.");

                if (endpointResponse.IsSuccessStatusCode)
                {
                    success = true;
                }
                else
                {
                    retries = 0;
                }
            }
            catch (HttpRequestException)
            {
                Console.WriteLine("Connection error, retrying...");
                retries--;
                Thread.Sleep(10000);
            }
        }

        return success;
    }

    /// <summary>
    /// Delete a load balancer by its specified name.
    /// </summary>
    /// <param name="name">The name of the load balancer to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteLoadBalancerByName(string name)
    {
        try
        {
            var describeLoadBalancerResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                    new DescribeLoadBalancersRequest()
                    {
                        Names = new List<string>() { name }
                    });
            var lbArn =
describeLoadBalancerResponse.LoadBalancers[0].LoadBalancerArn;
            await _amazonElasticLoadBalancingV2.DeleteLoadBalancerAsync(
                new DeleteLoadBalancerRequest()
                {
                    LoadBalancerArn = lbArn
                }
```

```
            );
        }
        catch (LoadBalancerNotFoundException)
        {
            Console.WriteLine($"Load balancer {name} not found.");
        }
    }

    /// <summary>
    /// Delete a TargetGroup by its specified name.
    /// </summary>
    /// <param name="groupName">Name of the group to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteTargetGroupByName(string groupName)
    {
        var done = false;
        while (!done)
        {
            try
            {
                var groupResponse =
                    await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                        new DescribeTargetGroupsRequest()
                        {
                            Names = new List<string>() { groupName }
                        });

                var targetArn = groupResponse.TargetGroups[0].TargetGroupArn;
                await _amazonElasticLoadBalancingV2.DeleteTargetGroupAsync(
                    new DeleteTargetGroupRequest() { TargetGroupArn = targetArn });
                Console.WriteLine($"Deleted load balancing target group
{groupName}.");
                done = true;
            }
            catch (TargetGroupNotFoundException)
            {
                Console.WriteLine(
                    $"Target group {groupName} not found, could not delete.");
                done = true;
            }
            catch (ResourceInUseException)
            {
                Console.WriteLine("Target group not yet released, waiting...");
                Thread.Sleep(10000);
```

```
            }
        }
    }
}
```

DynamoDB を使用してレコメンデーションサービスをシミュレートするクラスを作成します。

```
/// <summary>
/// Encapsulates a DynamoDB table to use as a service that recommends books, movies,
 and songs.
/// </summary>
public class Recommendations
{
    private readonly IAmazonDynamoDB _amazonDynamoDb;
    private readonly DynamoDBContext _context;
    private readonly string _tableName;

    public string TableName => _tableName;

    /// <summary>
    /// Constructor for the Recommendations service.
    /// </summary>
    /// <param name="amazonDynamoDb">The injected DynamoDb client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public Recommendations(IAmazonDynamoDB amazonDynamoDb, IConfiguration
 configuration)
    {
        _amazonDynamoDb = amazonDynamoDb;
        _context = new DynamoDBContext(_amazonDynamoDb);
        _tableName = configuration["databaseName"]!;
    }

    /// <summary>
    /// Create the DynamoDb table with a specified name.
    /// </summary>
    /// <param name="tableName">The name for the table.</param>
    /// <returns>True when ready.</returns>
    public async Task<bool> CreateDatabaseWithName(string tableName)
    {
        try
        {
            Console.Write($"Creating table {tableName}...");
```

```
            var createRequest = new CreateTableRequest()
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                    {
                        new AttributeDefinition()
                        {
                            AttributeName = "MediaType",
                            AttributeType = ScalarAttributeType.S
                        },
                        new AttributeDefinition()
                        {
                            AttributeName = "ItemId",
                            AttributeType = ScalarAttributeType.N
                        }
                    },
                KeySchema = new List<KeySchemaElement>()
                    {
                        new KeySchemaElement()
                        {
                            AttributeName = "MediaType",
                            KeyType = KeyType.HASH
                        },
                        new KeySchemaElement()
                        {
                            AttributeName = "ItemId",
                            KeyType = KeyType.RANGE
                        }
                    },
                ProvisionedThroughput = new ProvisionedThroughput()
                {
                    ReadCapacityUnits = 5,
                    WriteCapacityUnits = 5
                }
            };
            await _amazonDynamoDb.CreateTableAsync(createRequest);

            // Wait until the table is ACTIVE and then report success.
            Console.Write("\nWaiting for table to become active...");

            var request = new DescribeTableRequest
            {
                TableName = tableName
            };
```

```
            TableStatus status;
            do
            {
                Thread.Sleep(2000);

                var describeTableResponse = await
 _amazonDynamoDb.DescribeTableAsync(request);
                status = describeTableResponse.Table.TableStatus;

                Console.Write(".");
            }
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException)
        {
            Console.WriteLine($"Table {tableName} already exists.");
            return false;
        }
    }

    /// <summary>
    /// Populate the database table with data from a specified path.
    /// </summary>
    /// <param name="databaseTableName">The name of the table.</param>
    /// <param name="recommendationsPath">The path of the recommendations data.</
param>
    /// <returns>Async task.</returns>
    public async Task PopulateDatabase(string databaseTableName, string
 recommendationsPath)
    {
        var recommendationsText = await File.ReadAllTextAsync(recommendationsPath);
        var records =
            JsonSerializer.Deserialize<RecommendationModel[]>(recommendationsText);
        var batchWrite = _context.CreateBatchWrite<RecommendationModel>();

        foreach (var record in records!)
        {
            batchWrite.AddPutItem(record);
        }

        await batchWrite.ExecuteAsync();
```

```
    }

    /// <summary>
    /// Delete the recommendation table by name.
    /// </summary>
    /// <param name="tableName">The name of the recommendation table.</param>
    /// <returns>Async task.</returns>
    public async Task DestroyDatabaseByName(string tableName)
    {
        try
        {
            await _amazonDynamoDb.DeleteTableAsync(
                new DeleteTableRequest() { TableName = tableName });
            Console.WriteLine($"Table {tableName} was deleted.");
        }
        catch (ResourceNotFoundException)
        {
            Console.WriteLine($"Table {tableName} not found");
        }
    }
}
```

Systems Manager のアクションをラップするクラスを作成します。

```
/// <summary>
/// Encapsulates Systems Manager parameter operations. This example uses these
 parameters
/// to drive the demonstration of resilient architecture, such as failure of a
 dependency or
/// how the service responds to a health check.
/// </summary>
public class SmParameterWrapper
{
    private readonly IAmazonSimpleSystemsManagement _amazonSimpleSystemsManagement;

    private readonly string _tableParameter = "doc-example-resilient-architecture-
table";
    private readonly string _failureResponseParameter = "doc-example-resilient-
architecture-failure-response";
    private readonly string _healthCheckParameter = "doc-example-resilient-
architecture-health-check";
    private readonly string _tableName = "";
```

```
    public string TableParameter => _tableParameter;
    public string TableName => _tableName;
    public string HealthCheckParameter => _healthCheckParameter;
    public string FailureResponseParameter => _failureResponseParameter;

    /// <summary>
    /// Constructor for the SmParameterWrapper.
    /// </summary>
    /// <param name="amazonSimpleSystemsManagement">The injected Simple Systems
 Management client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public SmParameterWrapper(IAmazonSimpleSystemsManagement
 amazonSimpleSystemsManagement, IConfiguration configuration)
    {
        _amazonSimpleSystemsManagement = amazonSimpleSystemsManagement;
        _tableName = configuration["databaseName"]!;
    }

    /// <summary>
    /// Reset the Systems Manager parameters to starting values for the demo.
    /// </summary>
    /// <returns>Async task.</returns>
    public async Task Reset()
    {
        await this.PutParameterByName(_tableParameter, _tableName);
        await this.PutParameterByName(_failureResponseParameter, "none");
        await this.PutParameterByName(_healthCheckParameter, "shallow");
    }

    /// <summary>
    /// Set the value of a named Systems Manager parameter.
    /// </summary>
    /// <param name="name">The name of the parameter.</param>
    /// <param name="value">The value to set.</param>
    /// <returns>Async task.</returns>
    public async Task PutParameterByName(string name, string value)
    {
        await _amazonSimpleSystemsManagement.PutParameterAsync(
            new PutParameterRequest() { Name = name, Value = value, Overwrite =
 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。
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer
  - CreateTargetGroup
  - DeleteAutoScalingGroup
  - DeleteInstanceProfile
  - DeleteLaunchTemplate
  - DeleteLoadBalancer
  - DeleteTargetGroup
  - DescribeAutoScalingGroups
  - DescribeAvailabilityZones
  - DescribeIamInstanceProfileAssociations
  - DescribeInstances
  - DescribeLoadBalancers
  - DescribeSubnets
  - DescribeTargetGroups
  - DescribeTargetHealth
  - DescribeVpcs
  - RebootInstances
  - ReplaceIamInstanceProfileAssociation
  - TerminateInstanceInAutoScalingGroup
  - UpdateAutoScalingGroup

グループとインスタンスを管理する

次のコードサンプルは、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から、リソースをクリーンアップします。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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.First().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 is not null)
        {
            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 is not null)
        {
            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!);
        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!);
```

```
        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。 Amazon EC2

```
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
        };

        var response = await
_amazonAutoScaling.CreateAutoScalingGroupAsync(request);
        Console.WriteLine($"{groupName} Auto Scaling Group created");
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }



    /// <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<Amazon.AutoScaling.Model.Activity>>
 DescribeScalingActivitiesAsync(
        string groupName)
    {
        var scalingActivitiesRequest = new DescribeScalingActivitiesRequest
        {
            AutoScalingGroupName = groupName,
            MaxRecords = 10,
        };

        var response = await
 _amazonAutoScaling.DescribeScalingActivitiesAsync(scalingActivitiesRequest);
        return response.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>();
        groups!.ForEach(group =>
        {
            if (group.AutoScalingGroupName == groupName)
            {
                group.Instances.ForEach(instance =>
                {
                    instanceIds.Add(instance.InstanceId);
```

```
                });
            }
        });

        var scalingGroupsRequest = new DescribeAutoScalingInstancesRequest
        {
            MaxRecords = 10,
            InstanceIds = instanceIds,
        };

        var response = await
 _amazonAutoScaling.DescribeAutoScalingInstancesAsync(scalingGroupsRequest);
        var 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 groupList = new List<string>
            {
                groupName,
            };

        var request = new DescribeAutoScalingGroupsRequest
        {
            AutoScalingGroupNames = groupList,
        };

        var response = await
 _amazonAutoScaling.DescribeAutoScalingGroupsAsync(request);
        var 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<Amazon.CloudWatch.Model.Metric>>
 GetCloudWatchMetricsAsync(string groupName)
    {
        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);

        return response.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 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);

        return response.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

を使用した Amazon Bedrock の例 AWS SDK for .NET

次のコード例は、Amazon Bedrock AWS SDK for .NET で 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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。ここでは GitHub、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。

開始方法

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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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 ListFoundationModelsExample
{
    /// <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
            {
                ListFoundationModelsResponse 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、「 API リファレンスListFoundationModels」の「」を参照してください。 AWS SDK for .NET

トピック

- アクション

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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Bedrock ファンデーションの利用可能な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
            {
                ListFoundationModelsResponse 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、「 API リファレンスListFoundationModels」の「」を参照してください。 AWS SDK for .NET

を使用した Amazon Bedrock ランタイムの例 AWS SDK for .NET

次のコード例は、Amazon Bedrock ランタイム AWS SDK for .NET で 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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。ここでは GitHub、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

AI21 ラボ Jurassic-2

会話

次のコード例は、Bedrock の Converse API を使用して、AI21 Labs Jurassic-2 にテキストメッセージを送信する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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、AI21 Labs Jurassic-2 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 AI21 Labs Jurassic-2.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.Collections.Generic;

// 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., Jurassic-2 Mid.
var modelId = "ai21.j2-mid-v1";

// 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、「 API リファレンス」のAWS SDK for .NET 「会話」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 AI21 Labs Jurassic-2 にテキストメッセージを送信する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

モデル呼び出し 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 AI21 Labs Jurassic-2.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.IO;
using System.Text.Json;
```

```
using System.Text.Json.Nodes;

// 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., Jurassic-2 Mid.
var modelId = "ai21.j2-mid-v1";

// 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
{
    prompt = userMessage,
    maxTokens = 512,
    temperature = 0.5
});

// 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["completions"]?[0]?["data"]?["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、「 API リファレンスInvokeModel」の「」を参照してください。 AWS SDK for .NET

Amazon Titan Text

会話

次のコード例は、Bedrock の Converse API を使用して Amazon Titan Text にテキストメッセージを送信する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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、Amazon Titan Text 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 Amazon Titan Text.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.Collections.Generic;

// 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., Titan Text Premier.
var modelId = "amazon.titan-text-premier-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、「 API リファレンス」のAWS SDK for .NET 「会話」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 Amazon Titan Text 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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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 Amazon Titan Text 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 Amazon Titan Text
// and print the response stream.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.Collections.Generic;
using System.Linq;

// 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., Titan Text Premier.
var modelId = "amazon.titan-text-premier-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、「 API リファレンスConverseStream」の「」を参照してください。AWS SDK for .NET

InvokeModel

次のコード例は、Invoke Model API を使用して Amazon Titan Text にテキストメッセージを送信する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

モデル呼び出し 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 Amazon Titan Text.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;

// 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., Titan Text Premier.
var modelId = "amazon.titan-text-premier-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
{
    inputText = userMessage,
    textGenerationConfig = new
    {
        maxTokenCount = 512,
        temperature = 0.5
```

```
    }
});

// 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["results"]?[0]?["outputText"] ?? "";
    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、「 API リファレンスInvokeModel」の「」を参照してください。 AWS SDK for .NET

InvokeModelWithResponseStream

次のコード例は、Invoke Model API を使用して Amazon Titan Text モデル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出力する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

モデル呼び出し API を使用して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 native inference API to send a text message to Amazon Titan Text
// and print the response stream.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;

// 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., Titan Text Premier.
var modelId = "amazon.titan-text-premier-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
{
    inputText = userMessage,
    textGenerationConfig = new
    {
        maxTokenCount = 512,
        temperature = 0.5
    }
});
```

```
// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelWithResponseStreamRequest()
{
    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 streamingResponse = await
 client.InvokeModelWithResponseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var item in streamingResponse.Body)
    {
        var chunk = JsonSerializer.Deserialize<JsonObject>((item as
 PayloadPart).Bytes);
        var text = chunk["outputText"] ?? "";
        Console.Write(text);
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- API の詳細については、「 API リファレンスInvokeModelWithResponseStream」の「」を参照してください。 AWS SDK for .NET

Anthropic Claude

会話

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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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 Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.Collections.Generic;

// 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、「 API リファレンス」のAWS SDK for .NET 「会話」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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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 Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.Collections.Generic;
using System.Linq;

// 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、「 API リファレンスConverseStream」の「」を参照してください。 AWS SDK for .NET

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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

モデル呼び出し 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 Amazon;
using Amazon.BedrockRuntime;
```

```
using Amazon.BedrockRuntime.Model;
using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;

// 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、「 API リファレンスInvokeModel」の「」を参照してください。 AWS SDK for .NET

InvokeModelWithResponseStream

次のコード例は、Invoke Model API を使用して Anthropic Claude モデル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出力する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

モデル呼び出し API を使用して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 native inference API to send a text message to Anthropic Claude
// and print the response stream.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;
```

```
// 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, the user message, and an inference
 configuration.
var request = new InvokeModelWithResponseStreamRequest()
{
    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 streamingResponse = await
 client.InvokeModelWithResponseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var item in streamingResponse.Body)
    {
        var chunk = JsonSerializer.Deserialize<JsonObject>((item as
 PayloadPart).Bytes);
        var text = chunk["delta"]?["text"] ?? "";
        Console.Write(text);
    }
```

```
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- API の詳細については、「 API リファレンスInvokeModelWithResponseStream」の「」を参照してください。 AWS SDK for .NET

Cohere Command

会話: すべてのモデル

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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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 コマンド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 Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.Collections.Generic;

// 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、「 API リファレンス」のAWS SDK for .NET 「会話」を参照してください。

ConverseStream: すべてのモデル

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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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 Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.Collections.Generic;
using System.Linq;

// 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、「 API リファレンスConverseStream」の「」を参照してください。AWS SDK for .NET

InvokeModel: コマンド R と R+

次のコード例は、Invoke Model API を使用して Cohere Command R と R+ にテキストメッセージを送信する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

モデル呼び出し 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 Cohere Command R.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;

// 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.";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    message = userMessage,
    max_tokens = 512,
    temperature = 0.5
});

// 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["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、「 API リファレンスInvokeModel」の「」を参照してください。 AWS SDK for .NET

InvokeModel: コマンドとコマンドライト

次のコード例は、Invoke Model API を使用して Cohere コマンドにテキストメッセージを送信する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

モデル呼び出し 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 Cohere Command.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;

// 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 Light.
var modelId = "cohere.command-light-text-v14";

// 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
{
    prompt = userMessage,
    max_tokens = 512,
    temperature = 0.5
});

// 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["generations"]?[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、「 API リファレンスInvokeModel」の「」を参照してください。 AWS SDK for .NET

InvokeModelWithResponseStream: コマンド R と R+

次のコード例は、レスポンスストリームでモデル呼び出し API を使用して、Cohere コマンドにテキストメッセージを送信する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

モデル呼び出し API を使用して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 native inference API to send a text message to Cohere Command R
// and print the response stream.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.IO;
using System.Text.Json;
```

```
using System.Text.Json.Nodes;

// 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.";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    message = userMessage,
    max_tokens = 512,
    temperature = 0.5
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelWithResponseStreamRequest()
{
    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 streamingResponse = await
 client.InvokeModelWithResponseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var item in streamingResponse.Body)
    {
        var chunk = JsonSerializer.Deserialize<JsonObject>((item as
 PayloadPart).Bytes);
        var text = chunk["text"] ?? "";
        Console.Write(text);
    }
}
catch (AmazonBedrockRuntimeException e)
{
```

```
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- API の詳細については、「 API リファレンスInvokeModel」の「」を参照してください。 AWS SDK for .NET

InvokeModelWithResponseStream: コマンドとコマンドライト

次のコード例は、レスポンスストリームでモデル呼び出し API を使用して、Cohere コマンドにテキストメッセージを送信する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

モデル呼び出し API を使用して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 native inference API to send a text message to Cohere Command
// and print the response stream.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;

// 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 Light.
var modelId = "cohere.command-light-text-v14";
```

```
// 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
{
    prompt = userMessage,
    max_tokens = 512,
    temperature = 0.5
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelWithResponseStreamRequest()
{
    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 streamingResponse = await
 client.InvokeModelWithResponseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var item in streamingResponse.Body)
    {
        var chunk = JsonSerializer.Deserialize<JsonObject>((item as
 PayloadPart).Bytes);
        var text = chunk["generations"]?[0]?["text"] ?? "";
        Console.Write(text);
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- API の詳細については、「 API リファレンスInvokeModel」の「」を参照してください。 AWS SDK for .NET

メタラマ

すべてのモデル: Converse API

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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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 Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.Collections.Generic;

// 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、「 API リファレンス」のAWS SDK for .NET 「会話」を参照してください。

ConverseStream: すべてのモデル

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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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 Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.Collections.Generic;
using System.Linq;

// 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、「 API リファレンスConverseStream」の「」を参照してください。 AWS SDK for .NET

InvokeModel: ラマ 2

次のコード例は、モデル呼び出し API を使用して Meta Llama 2 にテキストメッセージを送信する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

モデル呼び出し 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 Meta Llama 2.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;

// 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 2 Chat 13B.
var modelId = "meta.llama2-13b-chat-v1";

// Define the prompt for the model.
var prompt = "Describe the purpose of a 'hello world' program in one line.";

// Embed the prompt in Llama 2's instruction format.
var formattedPrompt = $"<s>[INST] {prompt} [/INST]";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    prompt = formattedPrompt,
    max_gen_len = 512,
    temperature = 0.5
});

// 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["generation"] ?? "";
    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、「 API リファレンスInvokeModel」の「」を参照してください。 AWS SDK for .NET

InvokeModel: ラマ 3

次のコード例は、Invoke Model API を使用して Meta Llama 3 にテキストメッセージを送信する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

モデル呼び出し 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 Meta Llama 3.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;

// 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 prompt for the model.
var prompt = "Describe the purpose of a 'hello world' program in one line.";

// Embed the prompt in Llama 2's instruction format.
var formattedPrompt = $@"
<|begin_of_text|>
<|start_header_id|>user<|end_header_id|>
{prompt}
<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    prompt = formattedPrompt,
    max_gen_len = 512,
    temperature = 0.5
});

// 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["generation"] ?? "";
    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、「 API リファレンスInvokeModel」の「」を参照してください。 AWS SDK for .NET

InvokeModelWithResponseStream: ラマ 2

次のコード例は、モデル呼び出し API を使用して Meta Llama 2 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出力する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

モデル呼び出し API を使用して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 native inference API to send a text message to Meta Llama 2
```

```
// and print the response stream.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;

// 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 2 Chat 13B.
var modelId = "meta.llama2-13b-chat-v1";

// Define the prompt for the model.
var prompt = "Describe the purpose of a 'hello world' program in one line.";

// Embed the prompt in Llama 2's instruction format.
var formattedPrompt = $"<s>[INST] {prompt} [/INST]";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    prompt = formattedPrompt,
    max_gen_len = 512,
    temperature = 0.5
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelWithResponseStreamRequest()
{
    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 streamingResponse = await
 client.InvokeModelWithResponseStreamAsync(request);
```

```
    // Extract and print the streamed response text in real-time.
    foreach (var item in streamingResponse.Body)
    {
        var chunk = JsonSerializer.Deserialize<JsonObject>((item as
 PayloadPart).Bytes);
        var text = chunk["generation"] ?? "";
        Console.Write(text);
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- API の詳細については、「 API リファレンスInvokeModelWithResponseStream」の「」を参照してください。 AWS SDK for .NET

InvokeModelWithResponseStream: ラマ 3

次のコード例は、モデル呼び出し API を使用して Meta Llama 3 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出力する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

モデル呼び出し API を使用して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 native inference API to send a text message to Meta Llama 3
// and print the response stream.

using Amazon;
using Amazon.BedrockRuntime;
```

```
using Amazon.BedrockRuntime.Model;
using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;

// 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 prompt for the model.
var prompt = "Describe the purpose of a 'hello world' program in one line.";

// Embed the prompt in Llama 2's instruction format.
var formattedPrompt = $@"
<|begin_of_text|>
<|start_header_id|>user<|end_header_id|>
{prompt}
<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    prompt = formattedPrompt,
    max_gen_len = 512,
    temperature = 0.5
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelWithResponseStreamRequest()
{
    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 streamingResponse = await
 client.InvokeModelWithResponseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var item in streamingResponse.Body)
    {
        var chunk = JsonSerializer.Deserialize<JsonObject>((item as
 PayloadPart).Bytes);
        var text = chunk["generation"] ?? "";
        Console.Write(text);
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- API の詳細については、「 API リファレンスInvokeModelWithResponseStream」の「」を参照してください。 AWS SDK for .NET

ミスタル AI

会話

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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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 Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.Collections.Generic;

// 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、「 API リファレンス」のAWS SDK for .NET 「会話」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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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 Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using Amazon.Runtime;
using System;
using System.Collections.Generic;
using System.Linq;

// 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、「 API リファレンスConverseStream」の「」を参照してください。 AWS SDK for .NET

InvokeModel

次のコード例は、Invoke Model API を使用して Mistral モデルにテキストメッセージを送信する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

モデル呼び出し 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 Mistral.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;

// 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 prompt for the model.
var prompt = "Describe the purpose of a 'hello world' program in one line.";

// Embed the prompt in Mistral's instruction format.
var formattedPrompt = $"<s>[INST] {prompt} [/INST]";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    prompt = formattedPrompt,
    max_tokens = 512,
    temperature = 0.5
});

// 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["outputs"]?[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、「 API リファレンスInvokeModel」の「」を参照してください。 AWS SDK for .NET

InvokeModelWithResponseStream

次のコード例は、Invoke Model API を使用して Mistral AI モデル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出力する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

モデル呼び出し API を使用して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 native inference API to send a text message to Mistral
// and print the response stream.

using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;

// 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 prompt for the model.
var prompt = "Describe the purpose of a 'hello world' program in one line.";

// Embed the prompt in Mistral's instruction format.
var formattedPrompt = $"<s>[INST] {prompt} [/INST]";
```

```
//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    prompt = formattedPrompt,
    max_tokens = 512,
    temperature = 0.5
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelWithResponseStreamRequest()
{
    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 streamingResponse = await
 client.InvokeModelWithResponseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var item in streamingResponse.Body)
    {
        var chunk = JsonSerializer.Deserialize<JsonObject>((item as
 PayloadPart).Bytes);
        var text = chunk["outputs"]?[0]?["text"] ?? "";
        Console.Write(text);
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- API の詳細については、「 API リファレンスInvokeModelWithResponseStream」の「」を参照してください。 AWS SDK for .NET

シナリオ

Amazon Bedrock 基盤モデルとやりとりするためのプレイグラウンドアプリケーションを作成します。

次のコード例は、さまざまな方法で Amazon Bedrock 基盤モデルと相互作用するプレイグラウンドを作成する方法を示しています。

AWS SDK for .NET

.NET 基盤モデル (FM) プレイグラウンドは、C# コードから Amazon Bedrock を使用する方法を紹介する.NET MAUI Blazor サンプルアプリケーションです。この例は、.NET 開発者と C# 開発者が Amazon Bedrock を使用してジェネレーティブな AI 対応アプリケーションを構築する方法を示しています。次の 4 つのプレイグラウンドを使用して Amazon Bedrock 基盤モデルをテストしたり操作したりできます。

- テキストプレイグラウンド。
- チャットプレイグラウンド。
- ボイスチャットプレイグラウンド。
- イメージプレイグラウンド。

この例には、アクセスできる基盤モデルとその特性も一覧表示されています。ソースコードとデプロイ手順については、「」の「 プロジェクト」を参照してくださいGitHub。

この例で使用されているサービス

- Amazon Bedrock ランタイム

AWS CloudFormation を使用した の例 AWS SDK for .NET

次のコード例は、 AWS SDK for .NET で 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 CloudFormation。

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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。このリンクには GitHub、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が記載されています。

開始方法

こんにち AWS CloudFormationは

次のコード例は、 AWS CloudFormationの使用を開始する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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;

    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;
        }
    }
}
```

- API の詳細については、「 API リファレンスDescribeStackResources」の「」を参照してください。 AWS SDK for .NET

CloudWatch を使用した の例 AWS SDK for .NET

次のコード例は、 AWS SDK for .NET で 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 CloudWatch。

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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。このリンクには GitHub、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が記載されています。

開始方法

こんにち CloudWatchは

次のコード例は、 の使用を開始する方法を示しています CloudWatch。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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($"Hello Amazon CloudWatch! Following are some metrics
 available in the {metricNamespace} namespace:");
        Console.WriteLine();
        foreach (var metric in response.Metrics.Take(5))
        {
            Console.WriteLine($"\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、「 API リファレンスListMetrics」の「」を参照してください。 AWS SDK for .NET

トピック

- アクション
- シナリオ

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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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、「 API リファレンスDeleteAlarms」の「」を参照してください。
  AWS SDK for .NET

`DeleteAnomalyDetector`

次の例は、DeleteAnomalyDetector を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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、「 API リファレンスDeleteAnomalyDetector」の「」を参照してください。 AWS SDK for .NET

`DeleteDashboards`

次の例は、`DeleteDashboards` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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、「 API リファレンスDeleteDashboards」の「」を参照してください。AWS SDK for .NET

`DescribeAlarmHistory`

次の例は、`DescribeAlarmHistory` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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、「 API リファレンスDescribeAlarmHistory」の「」を参照してください。 AWS SDK for .NET

`DescribeAlarms`

次の例は、`DescribeAlarms` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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、「 API リファレンスDescribeAlarms」の「」を参照してください。AWS SDK for .NET

`DescribeAlarmsForMetric`

次の例は、`DescribeAlarmsForMetric` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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;
    }
```

- API の詳細については、「 API リファレンスDescribeAlarmsForMetric」の「」を参照してください。 AWS SDK for .NET

`DescribeAnomalyDetectors`

次の例は、`DescribeAnomalyDetectors` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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、「 API リファレンスDescribeAnomalyDetectors」の「」を参照してください。 AWS SDK for .NET

`DisableAlarmActions`

次の例は、`DisableAlarmActions` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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 の詳細については、「 API リファレンスDisableAlarmActions」の「」を参照してください。 AWS SDK for .NET

`EnableAlarmActions`

次の例は、EnableAlarmActions を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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、「 API リファレンスEnableAlarmActions」の「」を参照してください。 AWS SDK for .NET

GetDashboard

次の例は、GetDashboard を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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、「 API リファレンスGetDashboard」の「」を参照してください。AWS SDK for .NET

GetMetricData

次の例は、GetMetricData を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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}";
        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,
            });

        await foreach (var data in paginatedMetricData.MetricDataResults)
        {
            metricData.Add(data);
        }
        return metricData;
    }
```

- API の詳細については、「 API リファレンスGetMetricData」の「」を参照してください。 AWS SDK for .NET

`GetMetricStatistics`

次の例は、`GetMetricStatistics` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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;
    }
```

- API の詳細については、「 API リファレンスGetMetricStatistics」の「」を参照してください。AWS SDK for .NET

GetMetricWidgetImage

次の例は、GetMetricWidgetImage を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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、「 API リファレンスGetMetricWidgetImage」の「」を参照してください。 AWS SDK for .NET

`ListDashboards`

次の例は、`ListDashboards` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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、「 API リファレンスListDashboards」の「」を参照してください。 AWS SDK for .NET

`ListMetrics`

次の例は、`ListMetrics` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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、「 API リファレンスListMetrics」の「」を参照してください。 AWS SDK for .NET

PutAnomalyDetector

次の例は、PutAnomalyDetector を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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、「 API リファレンスPutAnomalyDetector」の「」を参照してください。 AWS SDK for .NET

`PutDashboard`

次の例は、PutDashboard を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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;
    }
```

- API の詳細については、「 API リファレンスPutDashboard」の「」を参照してください。AWS SDK for .NET

`PutMetricAlarm`

次の例は、`PutMetricAlarm` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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、「 API リファレンスPutMetricAlarm」の「」を参照してください。 AWS SDK for .NET

`PutMetricData`

次の例は、PutMetricData を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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、「 API リファレンスPutMetricData」の「」を参照してください。 AWS SDK for .NET

シナリオ

メトリクス、ダッシュボード、およびアラームの使用を開始する

次のコードサンプルは、以下の操作方法を示しています。

- CloudWatch 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
- メトリクス画像の取得とリソースのクリーンアップ

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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の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++)
        {
            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;
    }
}
```

シナリオが CloudWatch アクションに使用するラッパーメソッド。

```
/// <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;
    }

    /// <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;
    }


    /// <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}";
        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,
            });

        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;
    }

    /// <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;
    }
}
```

- 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

CloudWatch を使用したログ記録の例 AWS SDK for .NET

次のコード例は、 を CloudWatch Logs AWS SDK for .NET で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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。このリンクには GitHub、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が記載されています。

トピック

- アクション

アクション

`AssociateKmsKey`

次の例は、`AssociateKmsKey` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

/// <summary>
/// Shows how to associate an AWS Key Management Service (AWS KMS) key with
/// an Amazon CloudWatch Logs log group.
/// </summary>
public class AssociateKmsKey
{
    public static async Task Main()
    {
        // This client object will be associated with the same AWS Region
        // as the default user on this system. If you need to use a
        // different AWS Region, pass it as a parameter to the client
        // constructor.
        var client = new AmazonCloudWatchLogsClient();
```

```
            string kmsKeyId = "arn:aws:kms:us-west-2:<account-
number>:key/7c9eccc2-38cb-4c4f-9db3-766ee8dd3ad4";
            string groupName = "cloudwatchlogs-example-loggroup";

            var request = new AssociateKmsKeyRequest
            {
                KmsKeyId = kmsKeyId,
                LogGroupName = groupName,
            };

            var response = await client.AssociateKmsKeyAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"Successfully associated KMS key ID: {kmsKeyId}
 with log group: {groupName}.");
            }
            else
            {
                Console.WriteLine("Could not make the association between:
 {kmsKeyId} and {groupName}.");
            }
        }
    }
```

- API の詳細については、「 API リファレンスAssociateKmsKey」の「」を参照してください。AWS SDK for .NET

`CancelExportTask`

次の例は、`CancelExportTask` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

/// <summary>
/// Shows how to cancel an Amazon CloudWatch Logs export task.
/// </summary>
public class CancelExportTask
{
    public static async Task Main()
    {
        // This client object will be associated with the same AWS Region
        // as the default user on this system. If you need to use a
        // different AWS Region, pass it as a parameter to the client
        // constructor.
        var client = new AmazonCloudWatchLogsClient();
        string taskId = "exampleTaskId";

        var request = new CancelExportTaskRequest
        {
            TaskId = taskId,
        };

        var response = await client.CancelExportTaskAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"{taskId} successfully canceled.");
        }
        else
        {
            Console.WriteLine($"{taskId} could not be canceled.");
        }
    }
}
```

- API の詳細については、「 API リファレンスCancelExportTask」の「」を参照してください。AWS SDK for .NET

CreateExportTask

次の例は、CreateExportTask を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

/// <summary>
/// Shows how to create an Export Task to export the contents of the Amazon
/// CloudWatch Logs to the specified Amazon Simple Storage Service (Amazon S3)
/// bucket.
/// </summary>
public class CreateExportTask
{
    public static async Task Main()
    {
        // This client object will be associated with the same AWS Region
        // as the default user on this system. If you need to use a
        // different AWS Region, pass it as a parameter to the client
        // constructor.
        var client = new AmazonCloudWatchLogsClient();
        string taskName = "export-task-example";
        string logGroupName = "cloudwatchlogs-example-loggroup";
        string destination = "doc-example-bucket";
        var fromTime = 1437584472382;
        var toTime = 1437584472833;

        var request = new CreateExportTaskRequest
        {
            From = fromTime,
            To = toTime,
            TaskName = taskName,
            LogGroupName = logGroupName,
```

```
                Destination = destination,
            };

            var response = await client.CreateExportTaskAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"The task, {taskName} with ID: " +
                                  $"{response.TaskId} has been created
successfully.");
            }
        }
    }
```

- API の詳細については、「 API リファレンスCreateExportTask」の「」を参照してください。 AWS SDK for .NET

`CreateLogGroup`

次の例は、`CreateLogGroup` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

    /// <summary>
    /// Shows how to create an Amazon CloudWatch Logs log group.
    /// </summary>
    public class CreateLogGroup
    {
```

```
        public static async Task Main()
        {
            // This client object will be associated with the same AWS Region
            // as the default user on this system. If you need to use a
            // different AWS Region, pass it as a parameter to the client
            // constructor.
            var client = new AmazonCloudWatchLogsClient();

            string logGroupName = "cloudwatchlogs-example-loggroup";

            var request = new CreateLogGroupRequest
            {
                LogGroupName = logGroupName,
            };

            var response = await client.CreateLogGroupAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"Successfully create log group with ID:
{logGroupName}.");
            }
            else
            {
                Console.WriteLine("Could not create log group.");
            }
        }
    }
```

- API の詳細については、「 API リファレンスCreateLogGroup」の「」を参照してください。AWS SDK for .NET

CreateLogStream

次の例は、CreateLogStream を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

/// <summary>
/// Shows how to create an Amazon CloudWatch Logs stream for a CloudWatch
/// log group.
/// </summary>
public class CreateLogStream
{
    public static async Task Main()
    {
        // This client object will be associated with the same AWS Region
        // as the default user on this system. If you need to use a
        // different AWS Region, pass it as a parameter to the client
        // constructor.
        var client = new AmazonCloudWatchLogsClient();
        string logGroupName = "cloudwatchlogs-example-loggroup";
        string logStreamName = "cloudwatchlogs-example-logstream";

        var request = new CreateLogStreamRequest
        {
            LogGroupName = logGroupName,
            LogStreamName = logStreamName,
        };

        var response = await client.CreateLogStreamAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"{logStreamName} successfully created for
{logGroupName}.");
        }
```

```
            else
            {
                Console.WriteLine("Could not create stream.");
            }
        }
    }
```

- API の詳細については、「 API リファレンスCreateLogStream」の「」を参照してください。 AWS SDK for .NET

`DeleteLogGroup`

次の例は、DeleteLogGroup を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

/// <summary>
/// Uses the Amazon CloudWatch Logs Service to delete an existing
/// CloudWatch Logs log group.
/// </summary>
public class DeleteLogGroup
{
    public static async Task Main()
    {
        var client = new AmazonCloudWatchLogsClient();
        string logGroupName = "cloudwatchlogs-example-loggroup";

        var request = new DeleteLogGroupRequest
```

```
            {
                LogGroupName = logGroupName,
            };

            var response = await client.DeleteLogGroupAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"Successfully deleted CloudWatch log group,
{logGroupName}.");
            }
        }
    }
```

- API の詳細については、「 API リファレンスDeleteLogGroup」の「」を参照してください。AWS SDK for .NET

`DescribeExportTasks`

次の例は、`DescribeExportTasks` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

    /// <summary>
    /// Shows how to retrieve a list of information about Amazon CloudWatch
    /// Logs export tasks.
    /// </summary>
    public class DescribeExportTasks
```

```
    {
        public static async Task Main()
        {
            // This client object will be associated with the same AWS Region
            // as the default user on this system. If you need to use a
            // different AWS Region, pass it as a parameter to the client
            // constructor.
            var client = new AmazonCloudWatchLogsClient();

            var request = new DescribeExportTasksRequest
            {
                Limit = 5,
            };

            var response = new DescribeExportTasksResponse();

            do
            {
                response = await client.DescribeExportTasksAsync(request);
                response.ExportTasks.ForEach(t =>
                {
                    Console.WriteLine($"{t.TaskName} with ID: {t.TaskId} has status:
{t.Status}");
                });
            }
            while (response.NextToken is not null);
        }
    }
```

- API の詳細については、「 API リファレンスDescribeExportTasks」の「」を参照してください。 AWS SDK for .NET

DescribeLogGroups

次の例は、DescribeLogGroups を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

/// <summary>
/// Retrieves information about existing Amazon CloudWatch Logs log groups
/// and displays the information on the console.
/// </summary>
public class DescribeLogGroups
{
    public static async Task Main()
    {
        // Creates a CloudWatch Logs client using the default
        // user. If you need to work with resources in another
        // AWS Region than the one defined for the default user,
        // pass the AWS Region as a parameter to the client constructor.
        var client = new AmazonCloudWatchLogsClient();

        bool done = false;
        string newToken = null;

        var request = new DescribeLogGroupsRequest
        {
            Limit = 5,
        };

        DescribeLogGroupsResponse response;

        do
        {
            if (newToken is not null)
            {
                request.NextToken = newToken;
```

```
                }

                response = await client.DescribeLogGroupsAsync(request);

                response.LogGroups.ForEach(lg =>
                {
                    Console.WriteLine($"{lg.LogGroupName} is associated with the
key: {lg.KmsKeyId}.");
                    Console.WriteLine($"Created on: {lg.CreationTime.Date.Date}");
                    Console.WriteLine($"Date for this group will be stored for:
{lg.RetentionInDays} days.\n");
                });

                if (response.NextToken is null)
                {
                    done = true;
                }
                else
                {
                    newToken = response.NextToken;
                }
            }
            while (!done);
        }
    }
```

- API の詳細については、「 API リファレンスDescribeLogGroups」の「」を参照してください。 AWS SDK for .NET

`StartLiveTail`

次の例は、`StartLiveTail` を使用する方法を説明しています。

AWS SDK for .NET

必要なファイルを含めます。

```
using Amazon;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;
```

Live Tail セッションを開始します。

```
            var client = new AmazonCloudWatchLogsClient();
            var request = new StartLiveTailRequest
            {
                LogGroupIdentifiers = logGroupIdentifiers,
                LogStreamNames = logStreamNames,
                LogEventFilterPattern = filterPattern,
            };

            var response = await client.StartLiveTailAsync(request);

            // Catch if request fails
            if (response.HttpStatusCode != System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine("Failed to start live tail session");
                return;
            }
```

Live Tail セッションのイベントは 2 つの方法で処理できます。

```
            /* Method 1
            * 1). Asynchronously loop through the event stream
            * 2). Set a timer to dispose the stream and stop the Live Tail session
at the end.
            */
            var eventStream = response.ResponseStream;
            var task = Task.Run(() =>
            {
                foreach (var item in eventStream)
                {
                    if (item is LiveTailSessionUpdate liveTailSessionUpdate)
                    {
                        foreach (var sessionResult in
liveTailSessionUpdate.SessionResults)
                        {
                            Console.WriteLine("Message : {0}",
sessionResult.Message);
                        }
                    }
```

```
                    if (item is LiveTailSessionStart)
                    {
                        Console.WriteLine("Live Tail session started");
                    }
                    // On-stream exceptions are processed here
                    if (item is CloudWatchLogsEventStreamException)
                    {
                        Console.WriteLine($"ERROR: {item}");
                    }
                }
            });
            // Close the stream to stop the session after a timeout
            if (!task.Wait(TimeSpan.FromSeconds(10))){
                eventStream.Dispose();
                Console.WriteLine("End of line");
            }
```

```
            /* Method 2
            * 1). Add event handlers to each event variable
            * 2). Start processing the stream and wait for a timeout using
AutoResetEvent
            */
            AutoResetEvent endEvent = new AutoResetEvent(false);
            var eventStream = response.ResponseStream;
            using (eventStream) // automatically disposes the stream to stop the
session after execution finishes
            {
                eventStream.SessionStartReceived += (sender, e) =>
                {
                    Console.WriteLine("LiveTail session started");
                };
                eventStream.SessionUpdateReceived += (sender, e) =>
                {
                    foreach (LiveTailSessionLogEvent logEvent in
e.EventStreamEvent.SessionResults){
                        Console.WriteLine("Message: {0}", logEvent.Message);
                    }
                };
                // On-stream exceptions are captured here
                eventStream.ExceptionReceived += (sender, e) =>
                {
                    Console.WriteLine($"ERROR: {e.EventStreamException.Message}");
```

```
            };

            eventStream.StartProcessing();
            // Stream events for this amount of time.
            endEvent.WaitOne(TimeSpan.FromSeconds(10));
            Console.WriteLine("End of line");
        }
```

- API の詳細については、「 API リファレンスStartLiveTail」の「」を参照してください。 AWS SDK for .NET

を使用した Amazon Cognito ID プロバイダーの例 AWS SDK for .NET

次のコード例は、Amazon Cognito ID プロバイダー AWS SDK for .NET で 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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。このリンクには GitHub、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が記載されています。

トピック

- アクション
- シナリオ

アクション

`AdminGetUser`

次の例は、`AdminGetUser` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get the specified user from an Amazon Cognito user pool with administrator
access.
    /// </summary>
    /// <param name="userName">The name of the user.</param>
    /// <param name="poolId">The Id of the Amazon Cognito user pool.</param>
    /// <returns>Async task.</returns>
    public async Task<UserStatusType> GetAdminUserAsync(string userName, string
poolId)
    {
        AdminGetUserRequest userRequest = new AdminGetUserRequest
        {
            Username = userName,
            UserPoolId = poolId,
        };

        var response = await _cognitoService.AdminGetUserAsync(userRequest);

        Console.WriteLine($"User status {response.UserStatus}");
        return response.UserStatus;
    }
```

- API の詳細については、「 API リファレンスAdminGetUser」の「」を参照してください。AWS SDK for .NET

`AdminInitiateAuth`

次の例は、`AdminInitiateAuth` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Initiate an admin auth request.
    /// </summary>
    /// <param name="clientId">The client ID to use.</param>
    /// <param name="userPoolId">The ID of the user pool.</param>
    /// <param name="userName">The username to authenticate.</param>
    /// <param name="password">The user's password.</param>
    /// <returns>The session to use in challenge-response.</returns>
    public async Task<string> AdminInitiateAuthAsync(string clientId, string
userPoolId, string userName, string password)
    {
        var authParameters = new Dictionary<string, string>();
        authParameters.Add("USERNAME", userName);
        authParameters.Add("PASSWORD", password);

        var request = new AdminInitiateAuthRequest
        {
            ClientId = clientId,
            UserPoolId = userPoolId,
            AuthParameters = authParameters,
            AuthFlow = AuthFlowType.ADMIN_USER_PASSWORD_AUTH,
        };

        var response = await _cognitoService.AdminInitiateAuthAsync(request);
        return response.Session;
    }
```

- API の詳細については、「 API リファレンスAdminInitiateAuth」の「」を参照してください。AWS SDK for .NET

AdminRespondToAuthChallenge

次の例は、AdminRespondToAuthChallenge を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Respond to an admin authentication challenge.
    /// </summary>
    /// <param name="userName">The name of the user.</param>
    /// <param name="clientId">The client ID.</param>
    /// <param name="mfaCode">The multi-factor authentication code.</param>
    /// <param name="session">The current application session.</param>
    /// <param name="clientId">The user pool ID.</param>
    /// <returns>The result of the authentication response.</returns>
    public async Task<AuthenticationResultType> AdminRespondToAuthChallengeAsync(
        string userName,
        string clientId,
        string mfaCode,
        string session,
        string userPoolId)
    {
        Console.WriteLine("SOFTWARE_TOKEN_MFA challenge is generated");

        var challengeResponses = new Dictionary<string, string>();
        challengeResponses.Add("USERNAME", userName);
        challengeResponses.Add("SOFTWARE_TOKEN_MFA_CODE", mfaCode);

        var respondToAuthChallengeRequest = new AdminRespondToAuthChallengeRequest
        {
            ChallengeName = ChallengeNameType.SOFTWARE_TOKEN_MFA,
            ClientId = clientId,
            ChallengeResponses = challengeResponses,
            Session = session,
            UserPoolId = userPoolId,
        };
```

```
        var response = await
_cognitoService.AdminRespondToAuthChallengeAsync(respondToAuthChallengeRequest);
        Console.WriteLine($"Response to Authentication
{response.AuthenticationResult.TokenType}");
        return response.AuthenticationResult;
    }
```

- API の詳細については、「 API リファレンスAdminRespondToAuthChallenge」の「」を参照してください。 AWS SDK for .NET

`AssociateSoftwareToken`

次の例は、`AssociateSoftwareToken` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get an MFA token to authenticate the user with the authenticator.
    /// </summary>
    /// <param name="session">The session name.</param>
    /// <returns>The session name.</returns>
    public async Task<string> AssociateSoftwareTokenAsync(string session)
    {
        var softwareTokenRequest = new AssociateSoftwareTokenRequest
        {
            Session = session,
        };

        var tokenResponse = await
_cognitoService.AssociateSoftwareTokenAsync(softwareTokenRequest);
        var secretCode = tokenResponse.SecretCode;
```

```
        Console.WriteLine($"Use the following secret code to set up the
authenticator: {secretCode}");

        return tokenResponse.Session;
    }
```

- API の詳細については、「 API リファレンスAssociateSoftwareToken」の「」を参照してください。 AWS SDK for .NET

`ConfirmDevice`

次の例は、ConfirmDevice を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Initiates and confirms tracking of the device.
    /// </summary>
    /// <param name="accessToken">The user's access token.</param>
    /// <param name="deviceKey">The key of the device from Amazon Cognito.</param>
    /// <param name="deviceName">The device name.</param>
    /// <returns></returns>
    public async Task<bool> ConfirmDeviceAsync(string accessToken, string deviceKey,
string deviceName)
    {
        var request = new ConfirmDeviceRequest
        {
            AccessToken = accessToken,
            DeviceKey = deviceKey,
            DeviceName = deviceName
        };

        var response = await _cognitoService.ConfirmDeviceAsync(request);
```

```
        return response.UserConfirmationNecessary;
    }
```

- API の詳細については、「 API リファレンスConfirmDevice」の「」を参照してください。 AWS SDK for .NET

`ConfirmSignUp`

次の例は、ConfirmSignUp を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Confirm that the user has signed up.
    /// </summary>
    /// <param name="clientId">The Id of this application.</param>
    /// <param name="code">The confirmation code sent to the user.</param>
    /// <param name="userName">The username.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> ConfirmSignupAsync(string clientId, string code, string
userName)
    {
        var signUpRequest = new ConfirmSignUpRequest
        {
            ClientId = clientId,
            ConfirmationCode = code,
            Username = userName,
        };

        var response = await _cognitoService.ConfirmSignUpAsync(signUpRequest);
        if (response.HttpStatusCode == HttpStatusCode.OK)
        {
            Console.WriteLine($"{userName} was confirmed");
```

```
            return true;
        }
        return false;
    }
```

- API の詳細については、「 API リファレンスConfirmSignUp」の「」を参照してください。 AWS SDK for .NET

`InitiateAuth`

次の例は、InitiateAuth を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Initiate authorization.
    /// </summary>
    /// <param name="clientId">The client Id of the application.</param>
    /// <param name="userName">The name of the user who is authenticating.</param>
    /// <param name="password">The password for the user who is authenticating.</
param>
    /// <returns>The response from the initiate auth request.</returns>
    public async Task<InitiateAuthResponse> InitiateAuthAsync(string clientId,
 string userName, string password)
    {
        var authParameters = new Dictionary<string, string>();
        authParameters.Add("USERNAME", userName);
        authParameters.Add("PASSWORD", password);

        var authRequest = new InitiateAuthRequest

        {
            ClientId = clientId,
```

```
            AuthParameters = authParameters,
            AuthFlow = AuthFlowType.USER_PASSWORD_AUTH,
        };

        var response = await _cognitoService.InitiateAuthAsync(authRequest);
        Console.WriteLine($"Result Challenge is : {response.ChallengeName}");

        return response;
    }
```

- API の詳細については、「 API リファレンスInitiateAuth」の「」を参照してください。 AWS SDK for .NET

`ListUserPools`

次の例は、`ListUserPools` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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、「 API リファレンスListUserPools」の「」を参照してください。 AWS SDK for .NET

`ListUsers`

次の例は、`ListUsers` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get a list of users for the Amazon Cognito user pool.
    /// </summary>
    /// <param name="userPoolId">The user pool ID.</param>
    /// <returns>A list of users.</returns>
    public async Task<List<UserType>> ListUsersAsync(string userPoolId)
    {
        var request = new ListUsersRequest
        {
            UserPoolId = userPoolId
        };

        var users = new List<UserType>();

        var usersPaginator = _cognitoService.Paginators.ListUsers(request);
        await foreach (var response in usersPaginator.Responses)
        {
            users.AddRange(response.Users);
        }
```

```
        return users;
    }
```

- API の詳細については、「 API リファレンスListUsers」の「」を参照してください。 AWS SDK for .NET

`ResendConfirmationCode`

次の例は、ResendConfirmationCode を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Send a new confirmation code to a user.
    /// </summary>
    /// <param name="clientId">The Id of the client application.</param>
    /// <param name="userName">The username of user who will receive the code.</
param>
    /// <returns>The delivery details.</returns>
    public async Task<CodeDeliveryDetailsType> ResendConfirmationCodeAsync(string
 clientId, string userName)
    {
        var codeRequest = new ResendConfirmationCodeRequest
        {
            ClientId = clientId,
            Username = userName,
        };

        var response = await
 _cognitoService.ResendConfirmationCodeAsync(codeRequest);

        Console.WriteLine($"Method of delivery is
 {response.CodeDeliveryDetails.DeliveryMedium}");
```

```
        return response.CodeDeliveryDetails;
    }
```

- API の詳細については、「 API リファレンスResendConfirmationCode」の「」を参照してください。 AWS SDK for .NET

`SignUp`

次の例は、SignUp を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Sign up a new user.
    /// </summary>
    /// <param name="clientId">The client Id of the application.</param>
    /// <param name="userName">The username to use.</param>
    /// <param name="password">The user's password.</param>
    /// <param name="email">The email address of the user.</param>
    /// <returns>A Boolean value indicating whether the user was confirmed.</
returns>
    public async Task<bool> SignUpAsync(string clientId, string userName, string
 password, string email)
    {
        var userAttrs = new AttributeType
        {
            Name = "email",
            Value = email,
        };

        var userAttrsList = new List<AttributeType>();
```

```
        userAttrsList.Add(userAttrs);

        var signUpRequest = new SignUpRequest
        {
            UserAttributes = userAttrsList,
            Username = userName,
            ClientId = clientId,
            Password = password
        };

        var response = await _cognitoService.SignUpAsync(signUpRequest);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスSignUp」の「」を参照してください。 AWS SDK for .NET

`VerifySoftwareToken`

次の例は、`VerifySoftwareToken` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Verify the TOTP and register for MFA.
    /// </summary>
    /// <param name="session">The name of the session.</param>
    /// <param name="code">The MFA code.</param>
    /// <returns>The status of the software token.</returns>
    public async Task<VerifySoftwareTokenResponseType>
VerifySoftwareTokenAsync(string session, string code)
    {
        var tokenRequest = new VerifySoftwareTokenRequest
```

```
        {
            UserCode = code,
            Session = session,
        };

        var verifyResponse = await
_cognitoService.VerifySoftwareTokenAsync(tokenRequest);

        return verifyResponse.Status;
    }
```

- API の詳細については、「 API リファレンスVerifySoftwareToken」の「」を参照してください。 AWS SDK for .NET

シナリオ

MFA を必要とするユーザープールによりユーザーをサインアップする

次のコードサンプルは、以下の操作方法を示しています。

- ユーザー名、パスワード、E メールアドレスでサインアップしてユーザーを確認します。
- MFA アプリケーションをユーザーに関連付けて、多要素認証を設定します。
- パスワードと MFA コードを使用してサインインします。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
namespace CognitoBasics;

public class CognitoBasics
{
    private static ILogger logger = null!;
```

```
   static async Task Main(string[] args)
   {
       // Set up dependency injection for Amazon Cognito.
       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<IAmazonCognitoIdentityProvider>()
           .AddTransient<CognitoWrapper>()
           )
           .Build();

       logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
           .CreateLogger<CognitoBasics>();

       var configuration = new ConfigurationBuilder()
           .SetBasePath(Directory.GetCurrentDirectory())
           .AddJsonFile("settings.json") // Load settings from .json file.
           .AddJsonFile("settings.local.json",
               true) // Optionally load local settings.
           .Build();

       var cognitoWrapper = host.Services.GetRequiredService<CognitoWrapper>();

       Console.WriteLine(new string('-', 80));
       UiMethods.DisplayOverview();
       Console.WriteLine(new string('-', 80));

       // clientId - The app client Id value that you get from the AWS CDK script.
       var clientId = configuration["ClientId"]; // "*** REPLACE WITH CLIENT ID
VALUE FROM CDK SCRIPT";

       // poolId - The pool Id that you get from the AWS CDK script.
       var poolId = configuration["PoolId"]!; // "*** REPLACE WITH POOL ID VALUE
FROM CDK SCRIPT";
       var userName = configuration["UserName"];
       var password = configuration["Password"];
       var email = configuration["Email"];

       // If the username wasn't set in the configuration file,
       // get it from the user now.
```

```
        if (userName is null)
        {
            do
            {
                Console.Write("Username: ");
                userName = Console.ReadLine();
            }
            while (string.IsNullOrEmpty(userName));
        }
        Console.WriteLine($"\nUsername: {userName}");

        // If the password wasn't set in the configuration file,
        // get it from the user now.
        if (password is null)
        {
            do
            {
                Console.Write("Password: ");
                password = Console.ReadLine();
            }
            while (string.IsNullOrEmpty(password));
        }

        // If the email address wasn't set in the configuration file,
        // get it from the user now.
        if (email is null)
        {
            do
            {
                Console.Write("Email: ");
                email = Console.ReadLine();
            } while (string.IsNullOrEmpty(email));
        }

        // Now sign up the user.
        Console.WriteLine($"\nSigning up {userName} with email address: {email}");
        await cognitoWrapper.SignUpAsync(clientId, userName, password, email);

        // Add the user to the user pool.
        Console.WriteLine($"Adding {userName} to the user pool");
        await cognitoWrapper.GetAdminUserAsync(userName, poolId);

        UiMethods.DisplayTitle("Get confirmation code");
        Console.WriteLine($"Conformation code sent to {userName}.");
```

```
        Console.Write("Would you like to send a new code? (Y/N) ");
        var answer = Console.ReadLine();

        if (answer!.ToLower() == "y")
        {
            await cognitoWrapper.ResendConfirmationCodeAsync(clientId, userName);
            Console.WriteLine("Sending a new confirmation code");
        }

        Console.Write("Enter confirmation code (from Email): ");
        var code = Console.ReadLine();

        await cognitoWrapper.ConfirmSignupAsync(clientId, code, userName);

        UiMethods.DisplayTitle("Checking status");
        Console.WriteLine($"Rechecking the status of {userName} in the user pool");
        await cognitoWrapper.GetAdminUserAsync(userName, poolId);

        Console.WriteLine($"Setting up authenticator for {userName} in the user
pool");
        var setupResponse = await cognitoWrapper.InitiateAuthAsync(clientId,
userName, password);

        var setupSession = await
cognitoWrapper.AssociateSoftwareTokenAsync(setupResponse.Session);
        Console.Write("Enter the 6-digit code displayed in Google Authenticator: ");
        var setupCode = Console.ReadLine();

        var setupResult = await
cognitoWrapper.VerifySoftwareTokenAsync(setupSession, setupCode);
        Console.WriteLine($"Setup status: {setupResult}");

        Console.WriteLine($"Now logging in {userName} in the user pool");
        var authSession = await cognitoWrapper.AdminInitiateAuthAsync(clientId,
poolId, userName, password);

        Console.Write("Enter a new 6-digit code displayed in Google Authenticator:
");
        var authCode = Console.ReadLine();

        var authResult = await
cognitoWrapper.AdminRespondToAuthChallengeAsync(userName, clientId, authCode,
authSession, poolId);
```

```
        Console.WriteLine($"Authenticated and received access token:
 {authResult.AccessToken}");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Cognito scenario is complete.");
        Console.WriteLine(new string('-', 80));
    }
}


using System.Net;

namespace CognitoActions;

/// <summary>
/// Methods to perform Amazon Cognito Identity Provider actions.
/// </summary>
public class CognitoWrapper
{
    private readonly IAmazonCognitoIdentityProvider _cognitoService;

    /// <summary>
    /// Constructor for the wrapper class containing Amazon Cognito actions.
    /// </summary>
    /// <param name="cognitoService">The Amazon Cognito client object.</param>
    public CognitoWrapper(IAmazonCognitoIdentityProvider cognitoService)
    {
        _cognitoService = cognitoService;
    }

    /// <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;
    }


    /// <summary>
    /// Get a list of users for the Amazon Cognito user pool.
    /// </summary>
    /// <param name="userPoolId">The user pool ID.</param>
    /// <returns>A list of users.</returns>
    public async Task<List<UserType>> ListUsersAsync(string userPoolId)
    {
        var request = new ListUsersRequest
        {
            UserPoolId = userPoolId
        };

        var users = new List<UserType>();

        var usersPaginator = _cognitoService.Paginators.ListUsers(request);
        await foreach (var response in usersPaginator.Responses)
        {
            users.AddRange(response.Users);
        }

        return users;
    }


    /// <summary>
    /// Respond to an admin authentication challenge.
    /// </summary>
    /// <param name="userName">The name of the user.</param>
    /// <param name="clientId">The client ID.</param>
    /// <param name="mfaCode">The multi-factor authentication code.</param>
    /// <param name="session">The current application session.</param>
    /// <param name="clientId">The user pool ID.</param>
    /// <returns>The result of the authentication response.</returns>
    public async Task<AuthenticationResultType> AdminRespondToAuthChallengeAsync(
        string userName,
        string clientId,
        string mfaCode,
        string session,
```

```
        string userPoolId)
    {
        Console.WriteLine("SOFTWARE_TOKEN_MFA challenge is generated");

        var challengeResponses = new Dictionary<string, string>();
        challengeResponses.Add("USERNAME", userName);
        challengeResponses.Add("SOFTWARE_TOKEN_MFA_CODE", mfaCode);

        var respondToAuthChallengeRequest = new AdminRespondToAuthChallengeRequest
        {
            ChallengeName = ChallengeNameType.SOFTWARE_TOKEN_MFA,
            ClientId = clientId,
            ChallengeResponses = challengeResponses,
            Session = session,
            UserPoolId = userPoolId,
        };

        var response = await
_cognitoService.AdminRespondToAuthChallengeAsync(respondToAuthChallengeRequest);
        Console.WriteLine($"Response to Authentication
{response.AuthenticationResult.TokenType}");
        return response.AuthenticationResult;
    }


    /// <summary>
    /// Verify the TOTP and register for MFA.
    /// </summary>
    /// <param name="session">The name of the session.</param>
    /// <param name="code">The MFA code.</param>
    /// <returns>The status of the software token.</returns>
    public async Task<VerifySoftwareTokenResponseType>
VerifySoftwareTokenAsync(string session, string code)
    {
        var tokenRequest = new VerifySoftwareTokenRequest
        {
            UserCode = code,
            Session = session,
        };

        var verifyResponse = await
_cognitoService.VerifySoftwareTokenAsync(tokenRequest);

        return verifyResponse.Status;
```

```
    }


    /// <summary>
    /// Get an MFA token to authenticate the user with the authenticator.
    /// </summary>
    /// <param name="session">The session name.</param>
    /// <returns>The session name.</returns>
    public async Task<string> AssociateSoftwareTokenAsync(string session)
    {
        var softwareTokenRequest = new AssociateSoftwareTokenRequest
        {
            Session = session,
        };

        var tokenResponse = await
_cognitoService.AssociateSoftwareTokenAsync(softwareTokenRequest);
        var secretCode = tokenResponse.SecretCode;

        Console.WriteLine($"Use the following secret code to set up the
authenticator: {secretCode}");

        return tokenResponse.Session;
    }


    /// <summary>
    /// Initiate an admin auth request.
    /// </summary>
    /// <param name="clientId">The client ID to use.</param>
    /// <param name="userPoolId">The ID of the user pool.</param>
    /// <param name="userName">The username to authenticate.</param>
    /// <param name="password">The user's password.</param>
    /// <returns>The session to use in challenge-response.</returns>
    public async Task<string> AdminInitiateAuthAsync(string clientId, string
userPoolId, string userName, string password)
    {
        var authParameters = new Dictionary<string, string>();
        authParameters.Add("USERNAME", userName);
        authParameters.Add("PASSWORD", password);

        var request = new AdminInitiateAuthRequest
        {
            ClientId = clientId,
```

```
            UserPoolId = userPoolId,
            AuthParameters = authParameters,
            AuthFlow = AuthFlowType.ADMIN_USER_PASSWORD_AUTH,
        };

        var response = await _cognitoService.AdminInitiateAuthAsync(request);
        return response.Session;
    }

    /// <summary>
    /// Initiate authorization.
    /// </summary>
    /// <param name="clientId">The client Id of the application.</param>
    /// <param name="userName">The name of the user who is authenticating.</param>
    /// <param name="password">The password for the user who is authenticating.</
param>
    /// <returns>The response from the initiate auth request.</returns>
    public async Task<InitiateAuthResponse> InitiateAuthAsync(string clientId,
 string userName, string password)
    {
        var authParameters = new Dictionary<string, string>();
        authParameters.Add("USERNAME", userName);
        authParameters.Add("PASSWORD", password);

        var authRequest = new InitiateAuthRequest

        {
            ClientId = clientId,
            AuthParameters = authParameters,
            AuthFlow = AuthFlowType.USER_PASSWORD_AUTH,
        };

        var response = await _cognitoService.InitiateAuthAsync(authRequest);
        Console.WriteLine($"Result Challenge is : {response.ChallengeName}");

        return response;
    }

    /// <summary>
    /// Confirm that the user has signed up.
    /// </summary>
    /// <param name="clientId">The Id of this application.</param>
    /// <param name="code">The confirmation code sent to the user.</param>
    /// <param name="userName">The username.</param>
```

```
    /// <returns>True if successful.</returns>
    public async Task<bool> ConfirmSignupAsync(string clientId, string code, string
userName)
    {
        var signUpRequest = new ConfirmSignUpRequest
        {
            ClientId = clientId,
            ConfirmationCode = code,
            Username = userName,
        };

        var response = await _cognitoService.ConfirmSignUpAsync(signUpRequest);
        if (response.HttpStatusCode == HttpStatusCode.OK)
        {
            Console.WriteLine($"{userName} was confirmed");
            return true;
        }
        return false;
    }


    /// <summary>
    /// Initiates and confirms tracking of the device.
    /// </summary>
    /// <param name="accessToken">The user's access token.</param>
    /// <param name="deviceKey">The key of the device from Amazon Cognito.</param>
    /// <param name="deviceName">The device name.</param>
    /// <returns></returns>
    public async Task<bool> ConfirmDeviceAsync(string accessToken, string deviceKey,
string deviceName)
    {
        var request = new ConfirmDeviceRequest
        {
            AccessToken = accessToken,
            DeviceKey = deviceKey,
            DeviceName = deviceName
        };

        var response = await _cognitoService.ConfirmDeviceAsync(request);
        return response.UserConfirmationNecessary;
    }


    /// <summary>
```

```
    /// Send a new confirmation code to a user.
    /// </summary>
    /// <param name="clientId">The Id of the client application.</param>
    /// <param name="userName">The username of user who will receive the code.</
param>
    /// <returns>The delivery details.</returns>
    public async Task<CodeDeliveryDetailsType> ResendConfirmationCodeAsync(string
 clientId, string userName)
    {
        var codeRequest = new ResendConfirmationCodeRequest
        {
            ClientId = clientId,
            Username = userName,
        };

        var response = await
 _cognitoService.ResendConfirmationCodeAsync(codeRequest);

        Console.WriteLine($"Method of delivery is
 {response.CodeDeliveryDetails.DeliveryMedium}");

        return response.CodeDeliveryDetails;
    }


    /// <summary>
    /// Get the specified user from an Amazon Cognito user pool with administrator
 access.
    /// </summary>
    /// <param name="userName">The name of the user.</param>
    /// <param name="poolId">The Id of the Amazon Cognito user pool.</param>
    /// <returns>Async task.</returns>
    public async Task<UserStatusType> GetAdminUserAsync(string userName, string
 poolId)
    {
        AdminGetUserRequest userRequest = new AdminGetUserRequest
        {
            Username = userName,
            UserPoolId = poolId,
        };

        var response = await _cognitoService.AdminGetUserAsync(userRequest);

        Console.WriteLine($"User status {response.UserStatus}");
```

```
        return response.UserStatus;
    }


    /// <summary>
    /// Sign up a new user.
    /// </summary>
    /// <param name="clientId">The client Id of the application.</param>
    /// <param name="userName">The username to use.</param>
    /// <param name="password">The user's password.</param>
    /// <param name="email">The email address of the user.</param>
    /// <returns>A Boolean value indicating whether the user was confirmed.</
returns>
    public async Task<bool> SignUpAsync(string clientId, string userName, string
 password, string email)
    {
        var userAttrs = new AttributeType
        {
            Name = "email",
            Value = email,
        };

        var userAttrsList = new List<AttributeType>();

        userAttrsList.Add(userAttrs);

        var signUpRequest = new SignUpRequest
        {
            UserAttributes = userAttrsList,
            Username = userName,
            ClientId = clientId,
            Password = password
        };

        var response = await _cognitoService.SignUpAsync(signUpRequest);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

}
```

- 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。
  - AdminGetUser
  - AdminInitiateAuth
  - AdminRespondToAuthChallenge
  - AssociateSoftwareToken
  - ConfirmDevice
  - ConfirmSignUp
  - InitiateAuth
  - ListUsers
  - ResendConfirmationCode
  - RespondToAuthChallenge
  - SignUp
  - VerifySoftwareToken

を使用した Amazon Comprehend の例 AWS SDK for .NET

次のコード例は、Amazon Comprehend AWS SDK for .NET で 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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。このリンクには GitHub、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が記載されています。

トピック

- アクション

アクション

`DetectDominantLanguage`

次の例は、DetectDominantLanguage を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

    /// <summary>
    /// This example calls the Amazon Comprehend service to determine the
    /// dominant language.
    /// </summary>
    public static class DetectDominantLanguage
    {
        /// <summary>
        /// Calls Amazon Comprehend to determine the dominant language used in
        /// the sample text.
        /// </summary>
        public static async Task Main()
        {
            string text = "It is raining today in Seattle.";

            var comprehendClient = new
AmazonComprehendClient(Amazon.RegionEndpoint.USWest2);

            Console.WriteLine("Calling DetectDominantLanguage\n");
            var detectDominantLanguageRequest = new DetectDominantLanguageRequest()
            {
                Text = text,
            };
```

```
            var detectDominantLanguageResponse = await
comprehendClient.DetectDominantLanguageAsync(detectDominantLanguageRequest);
            foreach (var dl in detectDominantLanguageResponse.Languages)
            {
                Console.WriteLine($"Language Code: {dl.LanguageCode}, Score:
{dl.Score}");
            }

            Console.WriteLine("Done");
        }
    }
```

- API の詳細については、「 API リファレンスDetectDominantLanguage」の「」を参照してください。 AWS SDK for .NET

`DetectEntities`

次の例は、`DetectEntities` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

/// <summary>
/// This example shows how to use the AmazonComprehend service detect any
/// entities in submitted text.
/// </summary>
public static class DetectEntities
{
    /// <summary>
```

```
        /// The main method calls the DetectEntitiesAsync method to find any
        /// entities in the sample code.
        /// </summary>
        public static async Task Main()
        {
            string text = "It is raining today in Seattle";

            var comprehendClient = new AmazonComprehendClient();

            Console.WriteLine("Calling DetectEntities\n");
            var detectEntitiesRequest = new DetectEntitiesRequest()
            {
                Text = text,
                LanguageCode = "en",
            };
            var detectEntitiesResponse = await
comprehendClient.DetectEntitiesAsync(detectEntitiesRequest);

            foreach (var e in detectEntitiesResponse.Entities)
            {
                Console.WriteLine($"Text: {e.Text}, Type: {e.Type}, Score:
{e.Score}, BeginOffset: {e.BeginOffset}, EndOffset: {e.EndOffset}");
            }

            Console.WriteLine("Done");
        }
    }
```

- API の詳細については、「 API リファレンスDetectEntities」の「」を参照してください。AWS SDK for .NET

`DetectKeyPhrases`

次の例は、DetectKeyPhrases を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

    /// <summary>
    /// This example shows how to use the Amazon Comprehend service to
    /// search text for key phrases.
    /// </summary>
    public static class DetectKeyPhrase
    {
        /// <summary>
        /// This method calls the Amazon Comprehend method DetectKeyPhrasesAsync
        /// to detect any key phrases in the sample text.
        /// </summary>
        public static async Task Main()
        {
            string text = "It is raining today in Seattle";

            var comprehendClient = new
AmazonComprehendClient(Amazon.RegionEndpoint.USWest2);

            // Call DetectKeyPhrases API
            Console.WriteLine("Calling DetectKeyPhrases");
            var detectKeyPhrasesRequest = new DetectKeyPhrasesRequest()
            {
                Text = text,
                LanguageCode = "en",
            };
            var detectKeyPhrasesResponse = await
comprehendClient.DetectKeyPhrasesAsync(detectKeyPhrasesRequest);
            foreach (var kp in detectKeyPhrasesResponse.KeyPhrases)
            {
```

```
                Console.WriteLine($"Text: {kp.Text}, Score: {kp.Score}, BeginOffset:
{kp.BeginOffset}, EndOffset: {kp.EndOffset}");
            }

            Console.WriteLine("Done");
        }
    }
```

- API の詳細については、「 API リファレンスDetectKeyPhrases」の「」を参照してください。AWS SDK for .NET

`DetectPiiEntities`

次の例は、`DetectPiiEntities` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

/// <summary>
/// This example shows how to use the Amazon Comprehend service to find
/// personally identifiable information (PII) within text submitted to the
/// DetectPiiEntitiesAsync method.
/// </summary>
public class DetectingPII
{
    /// <summary>
    /// This method calls the DetectPiiEntitiesAsync method to locate any
    /// personally dientifiable information within the supplied text.
    /// </summary>
```

```
        public static async Task Main()
        {
            var comprehendClient = new AmazonComprehendClient();
            var text = @"Hello Paul Santos. The latest statement for your
                         credit card account 1111-0000-1111-0000 was
                         mailed to 123 Any Street, Seattle, WA 98109.";

            var request = new DetectPiiEntitiesRequest
            {
                Text = text,
                LanguageCode = "EN",
            };

            var response = await comprehendClient.DetectPiiEntitiesAsync(request);

            if (response.Entities.Count > 0)
            {
                foreach (var entity in response.Entities)
                {
                    var entityValue = text.Substring(entity.BeginOffset,
entity.EndOffset - entity.BeginOffset);
                    Console.WriteLine($"{entity.Type}: {entityValue}");
                }
            }
        }
    }
```

- API の詳細については、「 API リファレンスDetectPiiEntities」の「」を参照してください。AWS SDK for .NET

DetectSentiment

次の例は、DetectSentiment を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

    /// <summary>
    /// This example shows how to detect the overall sentiment of the supplied
    /// text using the Amazon Comprehend service.
    /// </summary>
    public static class DetectSentiment
    {
        /// <summary>
        /// This method calls the DetetectSentimentAsync method to analyze the
        /// supplied text and determine the overal sentiment.
        /// </summary>
        public static async Task Main()
        {
            string text = "It is raining today in Seattle";

            var comprehendClient = new
AmazonComprehendClient(Amazon.RegionEndpoint.USWest2);

            // Call DetectKeyPhrases API
            Console.WriteLine("Calling DetectSentiment");
            var detectSentimentRequest = new DetectSentimentRequest()
            {
                Text = text,
                LanguageCode = "en",
            };
            var detectSentimentResponse = await
comprehendClient.DetectSentimentAsync(detectSentimentRequest);
            Console.WriteLine($"Sentiment: {detectSentimentResponse.Sentiment}");
            Console.WriteLine("Done");
        }
```

```
    }
```

- API の詳細については、「 API リファレンスDetectSentiment」の「」を参照してください。AWS SDK for .NET

`DetectSyntax`

次の例は、`DetectSyntax` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

/// <summary>
/// This example shows how to use Amazon Comprehend to detect syntax
/// elements by calling the DetectSyntaxAsync method.
/// </summary>
public class DetectingSyntax
{
    /// <summary>
    /// This method calls DetectSynaxAsync to identify the syntax elements
    /// in the sample text.
    /// </summary>
    public static async Task Main()
    {
        string text = "It is raining today in Seattle";

        var comprehendClient = new AmazonComprehendClient();

        // Call DetectSyntax API
```

```
            Console.WriteLine("Calling DetectSyntaxAsync\n");
            var detectSyntaxRequest = new DetectSyntaxRequest()
            {
                Text = text,
                LanguageCode = "en",
            };
            DetectSyntaxResponse detectSyntaxResponse = await
comprehendClient.DetectSyntaxAsync(detectSyntaxRequest);
            foreach (SyntaxToken s in detectSyntaxResponse.SyntaxTokens)
            {
                Console.WriteLine($"Text: {s.Text}, PartOfSpeech:
{s.PartOfSpeech.Tag}, BeginOffset: {s.BeginOffset}, EndOffset: {s.EndOffset}");
            }

            Console.WriteLine("Done");
        }
    }
```

- API の詳細については、「 API リファレンスDetectSyntax」の「」を参照してください。AWS SDK for .NET

`StartTopicsDetectionJob`

次の例は、`StartTopicsDetectionJob` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

    /// <summary>
```

```
    /// This example scans the documents in an Amazon Simple Storage Service
    /// (Amazon S3) bucket and analyzes it for topics. The results are stored
    /// in another bucket and then the resulting job properties are displayed
    /// on the screen. This example was created using the AWS SDK for .NEt
    /// version 3.7 and .NET Core version 5.0.
    /// </summary>
    public static class TopicModeling
    {
        /// <summary>
        /// This methos calls a topic detection job by calling the Amazon
        /// Comprehend StartTopicsDetectionJobRequest.
        /// </summary>
        public static async Task Main()
        {
            var comprehendClient = new AmazonComprehendClient();

            string inputS3Uri = "s3://input bucket/input path";
            InputFormat inputDocFormat = InputFormat.ONE_DOC_PER_FILE;
            string outputS3Uri = "s3://output bucket/output path";
            string dataAccessRoleArn = "arn:aws:iam::account ID:role/data access
role";
            int numberOfTopics = 10;

            var startTopicsDetectionJobRequest = new
StartTopicsDetectionJobRequest()
            {
                InputDataConfig = new InputDataConfig()
                {
                    S3Uri = inputS3Uri,
                    InputFormat = inputDocFormat,
                },
                OutputDataConfig = new OutputDataConfig()
                {
                    S3Uri = outputS3Uri,
                },
                DataAccessRoleArn = dataAccessRoleArn,
                NumberOfTopics = numberOfTopics,
            };

            var startTopicsDetectionJobResponse = await
comprehendClient.StartTopicsDetectionJobAsync(startTopicsDetectionJobRequest);

            var jobId = startTopicsDetectionJobResponse.JobId;
            Console.WriteLine("JobId: " + jobId);
```

```
            var describeTopicsDetectionJobRequest = new
DescribeTopicsDetectionJobRequest()
            {
                JobId = jobId,
            };

            var describeTopicsDetectionJobResponse = await
comprehendClient.DescribeTopicsDetectionJobAsync(describeTopicsDetectionJobRequest);

PrintJobProperties(describeTopicsDetectionJobResponse.TopicsDetectionJobProperties);

            var listTopicsDetectionJobsResponse = await
comprehendClient.ListTopicsDetectionJobsAsync(new
ListTopicsDetectionJobsRequest());
            foreach (var props in
listTopicsDetectionJobsResponse.TopicsDetectionJobPropertiesList)
            {
                PrintJobProperties(props);
            }
        }

        /// <summary>
        /// This method is a helper method that displays the job properties
        /// from the call to StartTopicsDetectionJobRequest.
        /// </summary>
        /// <param name="props">A list of properties from the call to
        /// StartTopicsDetectionJobRequest.</param>
        private static void PrintJobProperties(TopicsDetectionJobProperties props)
        {
            Console.WriteLine($"JobId: {props.JobId}, JobName: {props.JobName},
JobStatus: {props.JobStatus}");
            Console.WriteLine($"NumberOfTopics: {props.NumberOfTopics}\nInputS3Uri:
{props.InputDataConfig.S3Uri}");
            Console.WriteLine($"InputFormat: {props.InputDataConfig.InputFormat},
OutputS3Uri: {props.OutputDataConfig.S3Uri}");
        }
    }
```

- API の詳細については、「 API リファレンスStartTopicsDetectionJob」の「」を参照してください。 AWS SDK for .NET

を使用した DynamoDB の例 AWS SDK for .NET

次のコード例は、DynamoDB AWS SDK for .NET で 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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。このリンクには GitHub、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が記載されています。

開始方法

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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;

namespace DynamoDB_Actions;

public static class HelloDynamoDB
{
    static async Task Main(string[] args)
    {
        var dynamoDbClient = new AmazonDynamoDBClient();
```

```
        Console.WriteLine($"Hello Amazon Dynamo DB! Following are some of your
 tables:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get the first five tables.
        var response = await dynamoDbClient.ListTablesAsync(
            new ListTablesRequest()
            {
                Limit = 5
            });

        foreach (var table in response.TableNames)
        {
            Console.WriteLine($"\tTable: {table}");
            Console.WriteLine();
        }
    }
}
```

- API の詳細については、「 API リファレンスListTables」の「」を参照してください。 AWS SDK for .NET

トピック

- アクション
- シナリオ
- サーバーレスサンプル

アクション

`BatchExecuteStatement`

次の例は、BatchExecuteStatement を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

INSERT ステートメントのバッチを使用して項目を追加します。

```
        /// <summary>
        /// Inserts movies imported from a JSON file into the movie table by
        /// using an Amazon DynamoDB PartiQL INSERT statement.
        /// </summary>
        /// <param name="tableName">The name of the table into which the movie
        /// information will be inserted.</param>
        /// <param name="movieFileName">The name of the JSON file that contains
        /// movie information.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the insert operation.</returns>
        public static async Task<bool> InsertMovies(string tableName, string
movieFileName)
        {
            // Get the list of movies from the JSON file.
            var movies = ImportMovies(movieFileName);

            var success = false;

            if (movies is not null)
            {
                // Insert the movies in a batch using PartiQL. Because the
                // batch can contain a maximum of 25 items, insert 25 movies
                // at a time.
                string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";
                var statements = new List<BatchStatementRequest>();

                try
                {
                    for (var indexOffset = 0; indexOffset < 250; indexOffset += 25)
                    {
                        for (var i = indexOffset; i < indexOffset + 25; i++)
                        {
```

```
                            statements.Add(new BatchStatementRequest
                            {
                                Statement = insertBatch,
                                Parameters = new List<AttributeValue>
                                {
                                    new AttributeValue { S = movies[i].Title },
                                    new AttributeValue { N =
movies[i].Year.ToString() },
                                },
                            });
                        }

                        var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
                        {
                            Statements = statements,
                        });

                        // Wait between batches for movies to be successfully added.
                        System.Threading.Thread.Sleep(3000);

                        success = response.HttpStatusCode ==
System.Net.HttpStatusCode.OK;

                        // Clear the list of statements for the next batch.
                        statements.Clear();
                    }
                }
                catch (AmazonDynamoDBException ex)
                {
                    Console.WriteLine(ex.Message);
                }
            }

            return success;
        }

        /// <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 full path to the JSON file.</param>
        /// <returns>A generic list of movie objects.</returns>
        public static List<Movie> ImportMovies(string movieFileName)
```

```
        {
            if (!File.Exists(movieFileName))
            {
                return null!;
            }

            using var sr = new StreamReader(movieFileName);
            string json = sr.ReadToEnd();
            var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

            if (allMovies is not null)
            {
                // Return the first 250 entries.
                return allMovies.GetRange(0, 250);
            }
            else
            {
                return null!;
            }
        }
```

SELECT ステートメントのバッチを使用して項目を取得します。

```
        /// <summary>
        /// Gets movies from the movie table by
        /// using an Amazon DynamoDB PartiQL SELECT statement.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year1">The year of the first movie.</param>
        /// <param name="year2">The year of the second movie.</param>
        /// <returns>True if successful.</returns>
        public static async Task<bool> GetBatch(
            string tableName,
            string title1,
            string title2,
            int year1,
            int year2)
        {
            var getBatch = $"SELECT FROM {tableName} WHERE title = ? AND year = ?";
            var statements = new List<BatchStatementRequest>
```

```
            {
                new BatchStatementRequest
                {
                    Statement = getBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = getBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            if (response.Responses.Count > 0)
            {
                response.Responses.ForEach(r =>
                {
                    Console.WriteLine($"{r.Item["title"]}\t{r.Item["year"]}");
                });
                return true;
            }
            else
            {
                Console.WriteLine($"Couldn't find either {title1} or {title2}.");
                return false;
            }

        }
```

UPDATE ステートメントのバッチを使用して項目を更新します。

```
        /// <summary>
        /// Updates information for multiple movies.
        /// </summary>
        /// <param name="tableName">The name of the table containing the
        /// movies to be updated.</param>
        /// <param name="producer1">The producer name for the first movie
        /// to update.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="year1">The year that the first movie was released.</param>
        /// <param name="producer2">The producer name for the second
        /// movie to update.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year2">The year that the second movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the update.</
returns>
        public static async Task<bool> UpdateBatch(
            string tableName,
            string producer1,
            string title1,
            int year1,
            string producer2,
            string title2,
            int year2)
        {

            string updateBatch = $"UPDATE {tableName} SET Producer=? WHERE title = ?
 AND year = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = producer1 },
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },
```

```
                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = producer2 },
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
 BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

DELETE ステートメントのバッチを使用して項目を削除します。

```
        /// <summary>
        /// Deletes multiple movies using a PartiQL BatchExecuteAsync
        /// statement.
        /// </summary>
        /// <param name="tableName">The name of the table containing the
        /// moves that will be deleted.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="year1">The year the first movie was released.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year2">The year the second movie was released.</param>
        /// <returns>A Boolean value indicating the success of the operation.</
returns>
        public static async Task<bool> DeleteBatch(
            string tableName,
            string title1,
            int year1,
            string title2,
            int year2)
```

```
        {

            string updateBatch = $"DELETE FROM {tableName} WHERE title = ? AND year
= ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- API の詳細については、「 API リファレンスBatchExecuteStatement」の「」を参照してください。 AWS SDK for .NET

BatchGetItem

次の例は、BatchGetItem を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;

namespace LowLevelBatchGet
{
    public class LowLevelBatchGet
    {
        private static readonly string _table1Name = "Forum";
        private static readonly string _table2Name = "Thread";

        public static async void RetrieveMultipleItemsBatchGet(AmazonDynamoDBClient
 client)
        {
            var request = new BatchGetItemRequest
            {
                RequestItems = new Dictionary<string, KeysAndAttributes>()
            {
                { _table1Name,
                  new KeysAndAttributes
                  {
                      Keys = new List<Dictionary<string, AttributeValue> >()
                      {
                          new Dictionary<string, AttributeValue>()
                          {
                              { "Name", new AttributeValue {
                            S = "Amazon DynamoDB"
                        } }
                          },
                          new Dictionary<string, AttributeValue>()
                          {
                              { "Name", new AttributeValue {
                            S = "Amazon S3"
```

```
                        } }
                          }
                    }
                }},
            {
                _table2Name,
                new KeysAndAttributes
                {
                    Keys = new List<Dictionary<string, AttributeValue> >()
                    {
                        new Dictionary<string, AttributeValue>()
                        {
                            { "ForumName", new AttributeValue {
                                  S = "Amazon DynamoDB"
                              } },
                            { "Subject", new AttributeValue {
                                  S = "DynamoDB Thread 1"
                              } }
                        },
                        new Dictionary<string, AttributeValue>()
                        {
                            { "ForumName", new AttributeValue {
                                  S = "Amazon DynamoDB"
                              } },
                            { "Subject", new AttributeValue {
                                  S = "DynamoDB Thread 2"
                              } }
                        },
                        new Dictionary<string, AttributeValue>()
                        {
                            { "ForumName", new AttributeValue {
                                  S = "Amazon S3"
                              } },
                            { "Subject", new AttributeValue {
                                  S = "S3 Thread 1"
                              } }
                        }
                    }
                }
            }
        };

        BatchGetItemResponse response;
```

```
            do
            {
                Console.WriteLine("Making request");
                response = await client.BatchGetItemAsync(request);

                // Check the response.
                var responses = response.Responses; // Attribute list in the
response.

                foreach (var tableResponse in responses)
                {
                    var tableResults = tableResponse.Value;
                    Console.WriteLine("Items retrieved from table {0}",
tableResponse.Key);
                    foreach (var item1 in tableResults)
                    {
                        PrintItem(item1);
                    }
                }

                // Any unprocessed keys? could happen if you exceed
ProvisionedThroughput or some other error.
                Dictionary<string, KeysAndAttributes> unprocessedKeys =
response.UnprocessedKeys;
                foreach (var unprocessedTableKeys in unprocessedKeys)
                {
                    // Print table name.
                    Console.WriteLine(unprocessedTableKeys.Key);
                    // Print unprocessed primary keys.
                    foreach (var key in unprocessedTableKeys.Value.Keys)
                    {
                        PrintItem(key);
                    }
                }

                request.RequestItems = unprocessedKeys;
            } while (response.UnprocessedKeys.Count > 0);
        }

        private static void PrintItem(Dictionary<string, AttributeValue>
attributeList)
        {
            foreach (KeyValuePair<string, AttributeValue> kvp in attributeList)
            {
```

```
                string attributeName = kvp.Key;
                AttributeValue value = kvp.Value;

                Console.WriteLine(
                    attributeName + " " +
                    (value.S == null ? "" : "S=[" + value.S + "]") +
                    (value.N == null ? "" : "N=[" + value.N + "]") +
                    (value.SS == null ? "" : "SS=[" + string.Join(",",
 value.SS.ToArray()) + "]") +
                    (value.NS == null ? "" : "NS=[" + string.Join(",",
 value.NS.ToArray()) + "]")
                    );
            }
            Console.WriteLine("*************************************************");
        }

        static void Main()
        {
            var client = new AmazonDynamoDBClient();

            RetrieveMultipleItemsBatchGet(client);
        }
    }
}
```

- API の詳細については、「 API リファレンスBatchGetItem」の「」を参照してください。 AWS SDK for .NET

`BatchWriteItem`

次の例は、BatchWriteItem を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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 full path to the JSON file.</param>
        /// <returns>A generic list of movie objects.</returns>
        public static List<Movie> ImportMovies(string movieFileName)
        {
            if (!File.Exists(movieFileName))
            {
                return null;
            }

            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.
            return allMovies.GetRange(0, 250);
        }

        /// <summary>
        /// Writes 250 items to the movie table.
        /// </summary>
        /// <param name="client">The initialized DynamoDB client object.</param>
        /// <param name="movieFileName">A string containing the full path to
        /// the JSON file containing movie data.</param>
        /// <returns>A long integer value representing the number of movies
        /// imported from the JSON file.</returns>
        public static async Task<long> BatchWriteItemsAsync(
            AmazonDynamoDBClient client,
            string movieFileName)
        {
            var movies = ImportMovies(movieFileName);
            if (movies is null)
            {
                Console.WriteLine("Couldn't find the JSON file with movie data.");
```

```
                return 0;
            }

            var context = new DynamoDBContext(client);

            var movieBatch = context.CreateBatchWrite<Movie>();
            movieBatch.AddPutItems(movies);

            Console.WriteLine("Adding imported movies to the table.");
            await movieBatch.ExecuteAsync();

            return movies.Count;
        }
```

- API の詳細については、「 API リファレンスBatchWriteItem」の「」を参照してください。 AWS SDK for .NET

`CreateTable`

次の例は、CreateTable を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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="client">An initialized Amazon DynamoDB client object.</
param>
        /// <param name="tableName">The name of the table to create.</param>
        /// <returns>A Boolean value indicating the success of the operation.</
returns>
```

```
        public static async Task<bool> CreateMovieTableAsync(AmazonDynamoDBClient
client, string tableName)
        {
            var response = await client.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,
                    },
                },
                ProvisionedThroughput = new ProvisionedThroughput
                {
                    ReadCapacityUnits = 5,
                    WriteCapacityUnits = 5,
                },
            });

            // 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
            {
                System.Threading.Thread.Sleep(sleepDuration);

                var describeTableResponse = await
client.DescribeTableAsync(request);
                status = describeTableResponse.Table.TableStatus;

                Console.Write(".");
            }
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
```

- API の詳細については、「 API リファレンスCreateTable」の「」を参照してください。 AWS SDK for .NET

`DeleteItem`

次の例は、DeleteItem を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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="client">The initialized DynamoDB client object.</param>
        /// <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 static async Task<bool> DeleteItemAsync(
            AmazonDynamoDBClient client,
            string tableName,
            Movie movieToDelete)
        {
            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,
            };

            var response = await client.DeleteItemAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- API の詳細については、「 API リファレンスDeleteItem」の「」を参照してください。 AWS SDK for .NET

DeleteTable

次の例は、DeleteTable を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        public static async Task<bool> DeleteTableAsync(AmazonDynamoDBClient client,
string tableName)
        {
            var request = new DeleteTableRequest
            {
                TableName = tableName,
            };

            var response = await client.DeleteTableAsync(request);
            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"Table {response.TableDescription.TableName}
successfully deleted.");
                return true;
            }
            else
            {
                Console.WriteLine("Could not delete table.");
                return false;
            }
        }
```

- API の詳細については、「 API リファレンスDeleteTable」の「」を参照してください。 AWS SDK for .NET

`DescribeTable`

次の例は、`DescribeTable` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
private static async Task GetTableInformation()
{
    Console.WriteLine("\n*** Retrieving table information ***");

    var response = await Client.DescribeTableAsync(new DescribeTableRequest
    {
        TableName = ExampleTableName
    });

    var table = response.Table;
    Console.WriteLine($"Name: {table.TableName}");
    Console.WriteLine($"# of items: {table.ItemCount}");
    Console.WriteLine($"Provision Throughput (reads/sec): " +
                      $"{table.ProvisionedThroughput.ReadCapacityUnits}");
    Console.WriteLine($"Provision Throughput (writes/sec): " +
                      $"{table.ProvisionedThroughput.WriteCapacityUnits}");
}
```

- API の詳細については、「 API リファレンスDescribeTable」の「」を参照してください。AWS SDK for .NET

`ExecuteStatement`

次の例は、`ExecuteStatement` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

INSERT ステートメントを使用して項目を追加します。

```
        /// <summary>
        /// Inserts a single movie into the movies table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to insert.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the INSERT operation.</returns>
        public static async Task<bool> InsertSingleMovie(string tableName, string
movieTitle, int year)
        {
            string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertBatch,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

SELECT ステートメントを使用して項目を取得します。

```
        /// <summary>
        /// Uses a PartiQL SELECT statement to retrieve a single movie from the
        /// movie database.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="movieTitle">The title of the movie to retrieve.</param>
        /// <returns>A list of movie data. If no movie matches the supplied
```

```
        /// title, the list is empty.</returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
GetSingleMovie(string tableName, string movieTitle)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE title = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { S = movieTitle },
            };

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }
```

SELECT ステートメントを使用して項目の一覧を取得します。

```
        /// <summary>
        /// Retrieve multiple movies by year using a SELECT statement.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="year">The year the movies were released.</param>
        /// <returns></returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
GetMovies(string tableName, int year)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE year = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { N = year.ToString() },
            };

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
```

```
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }
```

UPDATE ステートメントを使用して項目を更新します。

```
        /// <summary>
        /// Updates a single movie in the table, adding information for the
        /// producer.
        /// </summary>
        /// <param name="tableName">the name of the table.</param>
        /// <param name="producer">The name of the producer.</param>
        /// <param name="movieTitle">The movie title.</param>
        /// <param name="year">The year the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// UPDATE operation.</returns>
        public static async Task<bool> UpdateSingleMovie(string tableName, string
producer, string movieTitle, int year)
        {
            string insertSingle = $"UPDATE {tableName} SET Producer=? WHERE title
= ? AND year = ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = producer },
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

DELETE ステートメントを使用して映画を 1 つ削除します。

```
        /// <summary>
        /// Deletes a single movie from the table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to delete.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// DELETE operation.</returns>
        public static async Task<bool> DeleteSingleMovie(string tableName, string
movieTitle, int year)
        {
            var deleteSingle = $"DELETE FROM {tableName} WHERE title = ? AND year
= ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = deleteSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- API の詳細については、「 API リファレンスExecuteStatement」の「」を参照してください。AWS SDK for .NET

GetItem

次の例は、GetItem を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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="client">An initialized Amazon DynamoDB client object.</
param>
        /// <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 static async Task<Dictionary<string, AttributeValue>>
 GetItemAsync(AmazonDynamoDBClient client, Movie newMovie, string tableName)
        {
            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 client.GetItemAsync(request);
            return response.Item;
        }
```

- API の詳細については、「 API リファレンスGetItem」の「」を参照してください。 AWS SDK for .NET

`ListTables`

次の例は、`ListTables` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
private static async Task ListMyTables()
{
    Console.WriteLine("\n*** Listing tables ***");

    string lastTableNameEvaluated = null;
    do
    {
        var response = await Client.ListTablesAsync(new ListTablesRequest
        {
            Limit = 2,
            ExclusiveStartTableName = lastTableNameEvaluated
        });

        foreach (var name in response.TableNames)
        {
            Console.WriteLine(name);
        }

        lastTableNameEvaluated = response.LastEvaluatedTableName;
    } while (lastTableNameEvaluated != null);
}
```

- API の詳細については、「 API リファレンスListTables」の「」を参照してください。 AWS SDK for .NET

PutItem

次の例は、PutItem を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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="client">An initialized Amazon DynamoDB client object.</
param>
        /// <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 static async Task<bool> PutItemAsync(AmazonDynamoDBClient client,
 Movie newMovie, string tableName)
        {
            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,
            };

            var response = await client.PutItemAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- API の詳細については、「 API リファレンスPutItem」の「」を参照してください。 AWS SDK for .NET

Query

次の例は、Query を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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="client">The initialized DynamoDB client object.</param>
        /// <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 static async Task<int> QueryMoviesAsync(AmazonDynamoDBClient client,
string tableName, int year)
        {
            var movieTable = Table.LoadTable(client, tableName);
            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;

        Search 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;
    }
```

- 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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        public static async Task<int> ScanTableAsync(
            AmazonDynamoDBClient client,
            string tableName,
            int startYear,
            int endYear)
        {
            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 client.ScanAsync(request);
                foundCount += response.Items.Count;
                response.Items.ForEach(i => DisplayItem(i));
                request.ExclusiveStartKey = response.LastEvaluatedKey;
```

```
            }
            while (response.LastEvaluatedKey.Count > 0);
            return foundCount;
        }
```

- 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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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="client">An initialized Amazon DynamoDB client object.</
param>
        /// <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 static async Task<bool> UpdateItemAsync(
            AmazonDynamoDBClient client,
            Movie newMovie,
            MovieInfo newInfo,
            string tableName)
```

```
        {
            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,
            };

            var response = await client.UpdateItemAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- API の詳細については、「 API リファレンスUpdateItem」の「」を参照してください。 AWS SDK for .NET

シナリオ

テーブル、項目、クエリで使用を開始する

次のコードサンプルは、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する。
- 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。
- 何年もの間にリリースされた映画をスキャンする。
- テーブルからムービーを削除し、テーブルを削除します。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
// This example application performs the following basic Amazon DynamoDB
// functions:
//
//     CreateTableAsync
//     PutItemAsync
//     UpdateItemAsync
//     BatchWriteItemAsync
//     GetItemAsync
//     DeleteItemAsync
//     Query
//     Scan
//     DeleteItemAsync
//
using Amazon.DynamoDBv2;
using DynamoDB_Actions;

public class DynamoDB_Basics
{
    // Separator for the console display.
    private static readonly string SepBar = new string('-', 80);

    public static async Task Main()
    {
        var client = new AmazonDynamoDBClient();
```

```
        var tableName = "movie_table";

        // Relative path to moviedata.json in the local repository.
        var movieFileName = @"..\..\..\..\..\..\..\resources\sample_files
\movies.json";

        DisplayInstructions();

        // Create a new table and wait for it to be active.
        Console.WriteLine($"Creating the new table: {tableName}");

        var success = await DynamoDbMethods.CreateMovieTableAsync(client,
 tableName);

        if (success)
        {
            Console.WriteLine($"\nTable: {tableName} successfully created.");
        }
        else
        {
            Console.WriteLine($"\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 DynamoDbMethods.PutItemAsync(client, 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 DynamoDbMethods.UpdateItemAsync(client, 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 DynamoDbMethods.BatchWriteItemsAsync(client,
movieFileName);
        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 DynamoDbMethods.GetItemAsync(client, lookupMovie,
tableName);
        if (item.Count > 0)
        {
```

```
            DynamoDbMethods.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 DynamoDbMethods.DeleteItemAsync(client, 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($"Movies released in {findYear}");
        var queryCount = await DynamoDbMethods.QueryMoviesAsync(client, tableName,
findYear);
        Console.WriteLine($"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 DynamoDbMethods.ScanTableAsync(client, tableName,
startYear, endYear);
```

```
        Console.WriteLine($"Found {scanCount} movies released between {startYear}
and {endYear}");

        WaitForEnter();

        // Delete the table.
        success = await DynamoDbMethods.DeleteTableAsync(client, tableName);

        if (success)
        {
            Console.WriteLine($"Successfully deleted {tableName}");
        }
        else
        {
            Console.WriteLine($"Could not delete {tableName}");
        }

        Console.WriteLine("The DynamoDB Basics example application is done.");

        WaitForEnter();
    }

    /// <summary>
    /// Displays the description of the application on the console.
    /// </summary>
    private static void DisplayInstructions()
    {
        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()
    {
        Console.WriteLine("\nPress <Enter> to continue.");
        Console.WriteLine(SepBar);
        _ = Console.ReadLine();
    }
}
```

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="client">An initialized Amazon DynamoDB client object.</
param>
        /// <param name="tableName">The name of the table to create.</param>
        /// <returns>A Boolean value indicating the success of the operation.</
returns>
        public static async Task<bool> CreateMovieTableAsync(AmazonDynamoDBClient
 client, string tableName)
        {
            var response = await client.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,
                },
            },
            ProvisionedThroughput = new ProvisionedThroughput
            {
                ReadCapacityUnits = 5,
                WriteCapacityUnits = 5,
            },
        });

        // 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
        {
```

```
                System.Threading.Thread.Sleep(sleepDuration);

                var describeTableResponse = await
 client.DescribeTableAsync(request);
                status = describeTableResponse.Table.TableStatus;

                Console.Write(".");
            }
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
```

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="client">An initialized Amazon DynamoDB client object.</
param>
        /// <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 static async Task<bool> PutItemAsync(AmazonDynamoDBClient client,
 Movie newMovie, string tableName)
        {
            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,
            };
```

```
            var response = await client.PutItemAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

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="client">An initialized Amazon DynamoDB client object.</
param>
        /// <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 static async Task<bool> UpdateItemAsync(
            AmazonDynamoDBClient client,
            Movie newMovie,
            MovieInfo newInfo,
            string tableName)
        {
            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,
            };

            var response = await client.UpdateItemAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

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="client">An initialized Amazon DynamoDB client object.</
param>
        /// <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 static async Task<Dictionary<string, AttributeValue>>
 GetItemAsync(AmazonDynamoDBClient client, Movie newMovie, string tableName)
        {
            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 client.GetItemAsync(request);
        return response.Item;
    }
```

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 full path to the JSON file.</param>
    /// <returns>A generic list of movie objects.</returns>
    public static List<Movie> ImportMovies(string movieFileName)
    {
        if (!File.Exists(movieFileName))
        {
            return null;
        }

        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.
        return allMovies.GetRange(0, 250);
    }

    /// <summary>
    /// Writes 250 items to the movie table.
```

```
        /// </summary>
        /// <param name="client">The initialized DynamoDB client object.</param>
        /// <param name="movieFileName">A string containing the full path to
        /// the JSON file containing movie data.</param>
        /// <returns>A long integer value representing the number of movies
        /// imported from the JSON file.</returns>
        public static async Task<long> BatchWriteItemsAsync(
            AmazonDynamoDBClient client,
            string movieFileName)
        {
            var movies = ImportMovies(movieFileName);
            if (movies is null)
            {
                Console.WriteLine("Couldn't find the JSON file with movie data.");
                return 0;
            }

            var context = new DynamoDBContext(client);

            var movieBatch = context.CreateBatchWrite<Movie>();
            movieBatch.AddPutItems(movies);

            Console.WriteLine("Adding imported movies to the table.");
            await movieBatch.ExecuteAsync();

            return movies.Count;
        }
```

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="client">The initialized DynamoDB client object.</param>
        /// <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 static async Task<bool> DeleteItemAsync(
            AmazonDynamoDBClient client,
            string tableName,
            Movie movieToDelete)
        {
            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,
            };

            var response = await client.DeleteItemAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

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="client">The initialized DynamoDB client object.</param>
        /// <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 static async Task<int> QueryMoviesAsync(AmazonDynamoDBClient client,
string tableName, int year)
        {
            var movieTable = Table.LoadTable(client, tableName);
            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;

            Search 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;
        }
```

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。

```
        public static async Task<int> ScanTableAsync(
            AmazonDynamoDBClient client,
            string tableName,
            int startYear,
            int endYear)
        {
```

```
            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 client.ScanAsync(request);
                foundCount += response.Items.Count;
                response.Items.ForEach(i => DisplayItem(i));
                request.ExclusiveStartKey = response.LastEvaluatedKey;
            }
            while (response.LastEvaluatedKey.Count > 0);
            return foundCount;
        }
```

ムービーテーブルを削除します。

```
        public static async Task<bool> DeleteTableAsync(AmazonDynamoDBClient client,
string tableName)
        {
            var request = new DeleteTableRequest
            {
                TableName = tableName,
```

```
            };

            var response = await client.DeleteTableAsync(request);
            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"Table {response.TableDescription.TableName}
successfully deleted.");
                return true;
            }
            else
            {
                Console.WriteLine("Could not delete table.");
                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

PartiQL ステートメントのバッチを使用してテーブルにクエリを実行する

次のコードサンプルは、以下の操作方法を示しています。

- 複数の SELECT ステートメントを実行して、項目のバッチを取得する。
- 複数の INSERT ステートメントを実行して、項目のバッチを追加する。

- 複数の UPDATE ステートメントを実行して、項目のバッチを更新する。
- 複数の DELETE ステートメントを実行して、項目のバッチを削除する。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
// Before you run this example, download 'movies.json' from
// https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
GettingStarted.Js.02.html,
// and put it in the same folder as the example.

// Separator for the console display.
var SepBar = new string('-', 80);
const string tableName = "movie_table";
const string movieFileName = "moviedata.json";

DisplayInstructions();

// Create the table and wait for it to be active.
Console.WriteLine($"Creating the movie table: {tableName}");

var success = await DynamoDBMethods.CreateMovieTableAsync(tableName);
if (success)
{
    Console.WriteLine($"Successfully created table: {tableName}.");
}

WaitForEnter();

// Add movie information to the table from moviedata.json. See the
// instructions at the top of this file to download the JSON file.
Console.WriteLine($"Inserting movies into the new table. Please wait...");
success = await PartiQLBatchMethods.InsertMovies(tableName, movieFileName);
if (success)
{
```

```
    Console.WriteLine("Movies successfully added to the table.");
}
else
{
    Console.WriteLine("Movies could not be added to the table.");
}

WaitForEnter();

// Update multiple movies by using the BatchExecute statement.
var title1 = "Star Wars";
var year1 = 1977;
var title2 = "Wizard of Oz";
var year2 = 1939;

Console.WriteLine($"Updating two movies with producer information: {title1} and
 {title2}.");
success = await PartiQLBatchMethods.GetBatch(tableName, title1, title2, year1,
 year2);
if (success)
{
    Console.WriteLine($"Successfully retrieved {title1} and {title2}.");
}
else
{
    Console.WriteLine("Select statement failed.");
}

WaitForEnter();

// Update multiple movies by using the BatchExecute statement.
var producer1 = "LucasFilm";
var producer2 = "MGM";

Console.WriteLine($"Updating two movies with producer information: {title1} and
 {title2}.");
success = await PartiQLBatchMethods.UpdateBatch(tableName, producer1, title1, year1,
 producer2, title2, year2);
if (success)
{
    Console.WriteLine($"Successfully updated {title1} and {title2}.");
}
else
{
```

```
    Console.WriteLine("Update failed.");
}

WaitForEnter();

// Delete multiple movies by using the BatchExecute statement.
Console.WriteLine($"Now we will delete {title1} and {title2} from the table.");
success = await PartiQLBatchMethods.DeleteBatch(tableName, title1, year1, title2,
 year2);

if (success)
{
    Console.WriteLine($"Deleted {title1} and {title2}");
}
else
{
    Console.WriteLine($"could not delete {title1} or {title2}");
}

WaitForEnter();

// DNow that the PartiQL Batch scenario is complete, delete the movie table.
success = await DynamoDBMethods.DeleteTableAsync(tableName);

if (success)
{
    Console.WriteLine($"Successfully deleted {tableName}");
}
else
{
    Console.WriteLine($"Could not delete {tableName}");
}

/// <summary>
/// Displays the description of the application on the console.
/// </summary>
void DisplayInstructions()
{
    Console.Clear();
    Console.WriteLine();
    Console.Write(new string(' ', 24));
    Console.WriteLine("DynamoDB PartiQL Basics Example");
    Console.WriteLine(SepBar);
```

```
    Console.WriteLine("This demo application shows the basics of using Amazon
 DynamoDB with the AWS SDK for");
    Console.WriteLine(".NET version 3.7 and .NET 6.");
    Console.WriteLine(SepBar);
    Console.WriteLine("Creates a table by using the CreateTable method.");
    Console.WriteLine("Gets multiple movies by using a PartiQL SELECT statement.");
    Console.WriteLine("Updates multiple movies by using the ExecuteBatch method.");
    Console.WriteLine("Deletes multiple movies by using a PartiQL DELETE
 statement.");
    Console.WriteLine("Cleans up the resources created for the demo by deleting the
 table.");
    Console.WriteLine(SepBar);

    WaitForEnter();
}

/// <summary>
/// Simple method to wait for the <Enter> key to be pressed.
/// </summary>
void WaitForEnter()
{
    Console.WriteLine("\nPress <Enter> to continue.");
    Console.Write(SepBar);
    _ = Console.ReadLine();
}
```

```
        /// <summary>
        /// Gets movies from the movie table by
        /// using an Amazon DynamoDB PartiQL SELECT statement.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year1">The year of the first movie.</param>
        /// <param name="year2">The year of the second movie.</param>
        /// <returns>True if successful.</returns>
        public static async Task<bool> GetBatch(
            string tableName,
            string title1,
            string title2,
            int year1,
            int year2)
        {
```

```
            var getBatch = $"SELECT FROM {tableName} WHERE title = ? AND year = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = getBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = getBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            if (response.Responses.Count > 0)
            {
                response.Responses.ForEach(r =>
                {
                    Console.WriteLine($"{r.Item["title"]}\t{r.Item["year"]}");
                });
                return true;
            }
            else
            {
                Console.WriteLine($"Couldn't find either {title1} or {title2}.");
                return false;
            }
```

```
        }

        /// <summary>
        /// Inserts movies imported from a JSON file into the movie table by
        /// using an Amazon DynamoDB PartiQL INSERT statement.
        /// </summary>
        /// <param name="tableName">The name of the table into which the movie
        /// information will be inserted.</param>
        /// <param name="movieFileName">The name of the JSON file that contains
        /// movie information.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the insert operation.</returns>
        public static async Task<bool> InsertMovies(string tableName, string
movieFileName)
        {
            // Get the list of movies from the JSON file.
            var movies = ImportMovies(movieFileName);

            var success = false;

            if (movies is not null)
            {
                // Insert the movies in a batch using PartiQL. Because the
                // batch can contain a maximum of 25 items, insert 25 movies
                // at a time.
                string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";
                var statements = new List<BatchStatementRequest>();

                try
                {
                    for (var indexOffset = 0; indexOffset < 250; indexOffset += 25)
                    {
                        for (var i = indexOffset; i < indexOffset + 25; i++)
                        {
                            statements.Add(new BatchStatementRequest
                            {
                                Statement = insertBatch,
                                Parameters = new List<AttributeValue>
                                {
                                    new AttributeValue { S = movies[i].Title },
                                    new AttributeValue { N =
movies[i].Year.ToString() },
                                },
```

```
                        });
                    }

                    var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
                    {
                        Statements = statements,
                    });

                    // Wait between batches for movies to be successfully added.
                    System.Threading.Thread.Sleep(3000);

                    success = response.HttpStatusCode ==
System.Net.HttpStatusCode.OK;

                    // Clear the list of statements for the next batch.
                    statements.Clear();
                }
            }
            catch (AmazonDynamoDBException ex)
            {
                Console.WriteLine(ex.Message);
            }
        }

        return success;
    }

    /// <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 full path to the JSON file.</param>
    /// <returns>A generic list of movie objects.</returns>
    public static List<Movie> ImportMovies(string movieFileName)
    {
        if (!File.Exists(movieFileName))
        {
            return null!;
        }

        using var sr = new StreamReader(movieFileName);
        string json = sr.ReadToEnd();
        var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);
```

```
            if (allMovies is not null)
            {
                // Return the first 250 entries.
                return allMovies.GetRange(0, 250);
            }
            else
            {
                return null!;
            }
        }

        /// <summary>
        /// Updates information for multiple movies.
        /// </summary>
        /// <param name="tableName">The name of the table containing the
        /// movies to be updated.</param>
        /// <param name="producer1">The producer name for the first movie
        /// to update.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="year1">The year that the first movie was released.</param>
        /// <param name="producer2">The producer name for the second
        /// movie to update.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year2">The year that the second movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the update.</
returns>
        public static async Task<bool> UpdateBatch(
            string tableName,
            string producer1,
            string title1,
            int year1,
            string producer2,
            string title2,
            int year2)
        {

            string updateBatch = $"UPDATE {tableName} SET Producer=? WHERE title = ?
 AND year = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = updateBatch,
```

```
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = producer1 },
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = producer2 },
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
 BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }

        /// <summary>
        /// Deletes multiple movies using a PartiQL BatchExecuteAsync
        /// statement.
        /// </summary>
        /// <param name="tableName">The name of the table containing the
        /// moves that will be deleted.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="year1">The year the first movie was released.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year2">The year the second movie was released.</param>
        /// <returns>A Boolean value indicating the success of the operation.</
returns>
        public static async Task<bool> DeleteBatch(
            string tableName,
            string title1,
```

```
            int year1,
            string title2,
            int year2)
        {

            string updateBatch = $"DELETE FROM {tableName} WHERE title = ? AND year
= ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- API の詳細については、「 API リファレンスBatchExecuteStatement」の「」を参照してください。 AWS SDK for .NET

PartiQL を使用してテーブルに対してクエリを実行する

次のコードサンプルは、以下の操作方法を示しています。

- SELECT ステートメントを実行して項目を取得する。
- INSERT 文を実行して項目を追加する。
- UPDATE ステートメントを使用して項目を更新する。
- DELETE ステートメントを実行して項目を削除する。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
namespace PartiQL_Basics_Scenario
{
    public class PartiQLMethods
    {
        private static readonly AmazonDynamoDBClient Client = new
 AmazonDynamoDBClient();


        /// <summary>
        /// Inserts movies imported from a JSON file into the movie table by
        /// using an Amazon DynamoDB PartiQL INSERT statement.
        /// </summary>
        /// <param name="tableName">The name of the table where the movie
        /// information will be inserted.</param>
        /// <param name="movieFileName">The name of the JSON file that contains
        /// movie information.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the insert operation.</returns>
        public static async Task<bool> InsertMovies(string tableName, string
 movieFileName)
        {
            // Get the list of movies from the JSON file.
            var movies = ImportMovies(movieFileName);
```

```
            var success = false;

            if (movies is not null)
            {
                // Insert the movies in a batch using PartiQL. Because the
                // batch can contain a maximum of 25 items, insert 25 movies
                // at a time.
                string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";
                var statements = new List<BatchStatementRequest>();

                try
                {
                    for (var indexOffset = 0; indexOffset < 250; indexOffset += 25)
                    {
                        for (var i = indexOffset; i < indexOffset + 25; i++)
                        {
                            statements.Add(new BatchStatementRequest
                            {
                                Statement = insertBatch,
                                Parameters = new List<AttributeValue>
                                {
                                    new AttributeValue { S = movies[i].Title },
                                    new AttributeValue { N =
movies[i].Year.ToString() },
                                },
                            });
                        }

                        var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
                        {
                            Statements = statements,
                        });

                        // Wait between batches for movies to be successfully added.
                        System.Threading.Thread.Sleep(3000);

                        success = response.HttpStatusCode ==
System.Net.HttpStatusCode.OK;

                        // Clear the list of statements for the next batch.
                        statements.Clear();
```

```
                }
            }
            catch (AmazonDynamoDBException ex)
            {
                Console.WriteLine(ex.Message);
            }
        }

        return success;
    }

    /// <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 full path to the JSON file.</param>
    /// <returns>A generic list of movie objects.</returns>
    public static List<Movie> ImportMovies(string movieFileName)
    {
        if (!File.Exists(movieFileName))
        {
            return null!;
        }

        using var sr = new StreamReader(movieFileName);
        string json = sr.ReadToEnd();
        var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

        if (allMovies is not null)
        {
            // Return the first 250 entries.
            return allMovies.GetRange(0, 250);
        }
        else
        {
            return null!;
        }
    }



    /// <summary>
    /// Uses a PartiQL SELECT statement to retrieve a single movie from the
    /// movie database.
```

```
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="movieTitle">The title of the movie to retrieve.</param>
        /// <returns>A list of movie data. If no movie matches the supplied
        /// title, the list is empty.</returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
GetSingleMovie(string tableName, string movieTitle)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE title = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { S = movieTitle },
            };

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }



        /// <summary>
        /// Retrieve multiple movies by year using a SELECT statement.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="year">The year the movies were released.</param>
        /// <returns></returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
GetMovies(string tableName, int year)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE year = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { N = year.ToString() },
            };

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
```

```
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }


        /// <summary>
        /// Inserts a single movie into the movies table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to insert.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the INSERT operation.</returns>
        public static async Task<bool> InsertSingleMovie(string tableName, string
movieTitle, int year)
        {
            string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertBatch,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }



        /// <summary>
        /// Updates a single movie in the table, adding information for the
        /// producer.
        /// </summary>
        /// <param name="tableName">the name of the table.</param>
        /// <param name="producer">The name of the producer.</param>
```

```
        /// <param name="movieTitle">The movie title.</param>
        /// <param name="year">The year the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// UPDATE operation.</returns>
        public static async Task<bool> UpdateSingleMovie(string tableName, string
producer, string movieTitle, int year)
        {
            string insertSingle = $"UPDATE {tableName} SET Producer=? WHERE title
= ? AND year = ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = producer },
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }



        /// <summary>
        /// Deletes a single movie from the table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to delete.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// DELETE operation.</returns>
        public static async Task<bool> DeleteSingleMovie(string tableName, string
movieTitle, int year)
        {
            var deleteSingle = $"DELETE FROM {tableName} WHERE title = ? AND year
= ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
```

```
                Statement = deleteSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }


        /// <summary>
        /// Displays the list of movies returned from a database query.
        /// </summary>
        /// <param name="items">The list of movie information to display.</param>
        private static void DisplayMovies(List<Dictionary<string, AttributeValue>>
 items)
        {
            if (items.Count > 0)
            {
                Console.WriteLine($"Found {items.Count} movies.");
                items.ForEach(item =>
 Console.WriteLine($"{item["year"].N}\t{item["title"].S}"));
            }
            else
            {
                Console.WriteLine($"Didn't find a movie that matched the supplied
 criteria.");
            }
        }


    }
}



        /// <summary>
        /// Uses a PartiQL SELECT statement to retrieve a single movie from the
        /// movie database.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="movieTitle">The title of the movie to retrieve.</param>
```

```
        /// <returns>A list of movie data. If no movie matches the supplied
        /// title, the list is empty.</returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
GetSingleMovie(string tableName, string movieTitle)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE title = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { S = movieTitle },
            };

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }



        /// <summary>
        /// Inserts a single movie into the movies table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to insert.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the INSERT operation.</returns>
        public static async Task<bool> InsertSingleMovie(string tableName, string
movieTitle, int year)
        {
            string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertBatch,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
```

```
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }



        /// <summary>
        /// Updates a single movie in the table, adding information for the
        /// producer.
        /// </summary>
        /// <param name="tableName">the name of the table.</param>
        /// <param name="producer">The name of the producer.</param>
        /// <param name="movieTitle">The movie title.</param>
        /// <param name="year">The year the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// UPDATE operation.</returns>
        public static async Task<bool> UpdateSingleMovie(string tableName, string
producer, string movieTitle, int year)
        {
            string insertSingle = $"UPDATE {tableName} SET Producer=? WHERE title
= ? AND year = ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = producer },
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }



        /// <summary>
        /// Deletes a single movie from the table.
```

```
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to delete.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// DELETE operation.</returns>
        public static async Task<bool> DeleteSingleMovie(string tableName, string
movieTitle, int year)
        {
            var deleteSingle = $"DELETE FROM {tableName} WHERE title = ? AND year
= ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = deleteSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- API の詳細については、「 API リファレンスExecuteStatement」の「」を参照してください。AWS SDK for .NET

ドキュメントモデルを使用する

次のコード例は、DynamoDB と AWS SDK のドキュメントモデルを使用して、作成、読み取り、更新、削除 (CRUD) およびバッチオペレーションを実行する方法を示しています。

詳細については、「ドキュメントモデル」を参照してください。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ドキュメントモデルを使用して CRUD オペレーションを実行します。

```
/// <summary>
/// Performs CRUD operations on an Amazon DynamoDB table.
/// </summary>
public class MidlevelItemCRUD
{
    public static async Task Main()
    {
        var tableName = "ProductCatalog";
        var sampleBookId = 555;

        var client = new AmazonDynamoDBClient();
        var productCatalog = LoadTable(client, tableName);

        await CreateBookItem(productCatalog, sampleBookId);
        RetrieveBook(productCatalog, sampleBookId);

        // Couple of sample updates.
        UpdateMultipleAttributes(productCatalog, sampleBookId);
        UpdateBookPriceConditionally(productCatalog, sampleBookId);

        // Delete.
        await DeleteBook(productCatalog, sampleBookId);
    }

    /// <summary>
    /// Loads the contents of a DynamoDB table.
    /// </summary>
    /// <param name="client">An initialized DynamoDB client object.</param>
    /// <param name="tableName">The name of the table to load.</param>
    /// <returns>A DynamoDB table object.</returns>
    public static Table LoadTable(IAmazonDynamoDB client, string tableName)
    {
```

```
            Table productCatalog = Table.LoadTable(client, tableName);
            return productCatalog;
        }

        /// <summary>
        /// Creates an example book item and adds it to the DynamoDB table
        /// ProductCatalog.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
        public static async Task CreateBookItem(Table productCatalog, int
sampleBookId)
        {
            Console.WriteLine("\n*** Executing CreateBookItem() ***");
            var book = new Document
            {
                ["Id"] = sampleBookId,
                ["Title"] = "Book " + sampleBookId,
                ["Price"] = 19.99,
                ["ISBN"] = "111-1111111111",
                ["Authors"] = new List<string> { "Author 1", "Author 2", "Author
3" },
                ["PageCount"] = 500,
                ["Dimensions"] = "8.5x11x.5",
                ["InPublication"] = new DynamoDBBool(true),
                ["InStock"] = new DynamoDBBool(false),
                ["QuantityOnHand"] = 0,
            };

            // Adds the book to the ProductCatalog table.
            await productCatalog.PutItemAsync(book);
        }

        /// <summary>
        /// Retrieves an item, a book, from the DynamoDB ProductCatalog table.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
        public static async void RetrieveBook(
          Table productCatalog,
          int sampleBookId)
        {
```

```
            Console.WriteLine("\n*** Executing RetrieveBook() ***");

            // Optional configuration.
            var config = new GetItemOperationConfig
            {
                AttributesToGet = new List<string> { "Id", "ISBN", "Title",
"Authors", "Price" },
                ConsistentRead = true,
            };

            Document document = await productCatalog.GetItemAsync(sampleBookId,
config);
            Console.WriteLine("RetrieveBook: Printing book retrieved...");
            PrintDocument(document);
        }

        /// <summary>
        /// Updates multiple attributes for a book and writes the changes to the
        /// DynamoDB table ProductCatalog.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
        public static async void UpdateMultipleAttributes(
          Table productCatalog,
          int sampleBookId)
        {
            Console.WriteLine("\nUpdating multiple attributes....");
            int partitionKey = sampleBookId;

            var book = new Document
            {
                ["Id"] = partitionKey,

                // List of attribute updates.
                // The following replaces the existing authors list.
                ["Authors"] = new List<string> { "Author x", "Author y" },
                ["newAttribute"] = "New Value",
                ["ISBN"] = null, // Remove it.
            };

            // Optional parameters.
            var config = new UpdateItemOperationConfig
            {
```

```
                // Gets updated item in response.
                ReturnValues = ReturnValues.AllNewAttributes,
            };

            Document updatedBook = await productCatalog.UpdateItemAsync(book,
config);
            Console.WriteLine("UpdateMultipleAttributes: Printing item after
updates ...");
            PrintDocument(updatedBook);
        }

        /// <summary>
        /// Updates a book item if it meets the specified criteria.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
        public static async void UpdateBookPriceConditionally(
          Table productCatalog,
          int sampleBookId)
        {
            Console.WriteLine("\n*** Executing UpdateBookPriceConditionally() ***");

            int partitionKey = sampleBookId;

            var book = new Document
            {
                ["Id"] = partitionKey,
                ["Price"] = 29.99,
            };

            // For conditional price update, creating a condition expression.
            var expr = new Expression
            {
                ExpressionStatement = "Price = :val",
            };
            expr.ExpressionAttributeValues[":val"] = 19.00;

            // Optional parameters.
            var config = new UpdateItemOperationConfig
            {
                ConditionalExpression = expr,
                ReturnValues = ReturnValues.AllNewAttributes,
            };
```

```
            Document updatedBook = await productCatalog.UpdateItemAsync(book,
config);
            Console.WriteLine("UpdateBookPriceConditionally: Printing item whose
price was conditionally updated");
            PrintDocument(updatedBook);
        }

        /// <summary>
        /// Deletes the book with the supplied Id value from the DynamoDB table
        /// ProductCatalog.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
        public static async Task DeleteBook(
          Table productCatalog,
          int sampleBookId)
        {
            Console.WriteLine("\n*** Executing DeleteBook() ***");

            // Optional configuration.
            var config = new DeleteItemOperationConfig
            {
                // Returns the deleted item.
                ReturnValues = ReturnValues.AllOldAttributes,
            };
            Document document = await productCatalog.DeleteItemAsync(sampleBookId,
config);
            Console.WriteLine("DeleteBook: Printing deleted just deleted...");

            PrintDocument(document);
        }

        /// <summary>
        /// Prints the information for the supplied DynamoDB document.
        /// </summary>
        /// <param name="updatedDocument">A DynamoDB document object.</param>
        public static void PrintDocument(Document updatedDocument)
        {
            if (updatedDocument is null)
            {
                return;
            }
```

```
            foreach (var attribute in updatedDocument.GetAttributeNames())
            {
                string stringValue = null;
                var value = updatedDocument[attribute];

                if (value is null)
                {
                    continue;
                }

                if (value is Primitive)
                {
                    stringValue = value.AsPrimitive().Value.ToString();
                }
                else if (value is PrimitiveList)
                {
                    stringValue = string.Join(",", (from primitive
                      in value.AsPrimitiveList().Entries
                                                    select
primitive.Value).ToArray());
                }

                Console.WriteLine($"{attribute} - {stringValue}", attribute,
stringValue);
            }
        }
    }
```

ドキュメントモデルを使用してバッチ書き込みオペレーションを実行します。

```
    /// <summary>
    /// Shows how to use mid-level Amazon DynamoDB API calls to perform batch
    /// operations.
    /// </summary>
    public class MidLevelBatchWriteItem
    {
        public static async Task Main()
        {
            IAmazonDynamoDB client = new AmazonDynamoDBClient();
```

```
            await SingleTableBatchWrite(client);
            await MultiTableBatchWrite(client);
        }

        /// <summary>
        /// Perform a batch operation on a single DynamoDB table.
        /// </summary>
        /// <param name="client">An initialized DynamoDB object.</param>
        public static async Task SingleTableBatchWrite(IAmazonDynamoDB client)
        {
            Table productCatalog = Table.LoadTable(client, "ProductCatalog");
            var batchWrite = productCatalog.CreateBatchWrite();

            var book1 = new Document
            {
                ["Id"] = 902,
                ["Title"] = "My book1 in batch write using .NET helper classes",
                ["ISBN"] = "902-11-11-1111",
                ["Price"] = 10,
                ["ProductCategory"] = "Book",
                ["Authors"] = new List<string> { "Author 1", "Author 2", "Author
3" },
                ["Dimensions"] = "8.5x11x.5",
                ["InStock"] = new DynamoDBBool(true),
                ["QuantityOnHand"] = new DynamoDBNull(), // Quantity is unknown at
this time.
            };

            batchWrite.AddDocumentToPut(book1);

            // Specify delete item using overload that takes PK.
            batchWrite.AddKeyToDelete(12345);
            Console.WriteLine("Performing batch write in SingleTableBatchWrite()");
            await batchWrite.ExecuteAsync();
        }

        /// <summary>
        /// Perform a batch operation involving multiple DynamoDB tables.
        /// </summary>
        /// <param name="client">An initialized DynamoDB client object.</param>
        public static async Task MultiTableBatchWrite(IAmazonDynamoDB client)
        {
            // Specify item to add in the Forum table.
```

```
            Table forum = Table.LoadTable(client, "Forum");
            var forumBatchWrite = forum.CreateBatchWrite();

            var forum1 = new Document
            {
                ["Name"] = "Test BatchWrite Forum",
                ["Threads"] = 0,
            };
            forumBatchWrite.AddDocumentToPut(forum1);

            // Specify item to add in the Thread table.
            Table thread = Table.LoadTable(client, "Thread");
            var threadBatchWrite = thread.CreateBatchWrite();

            var thread1 = new Document
            {
                ["ForumName"] = "S3 forum",
                ["Subject"] = "My sample question",
                ["Message"] = "Message text",
                ["KeywordTags"] = new List<string> { "S3", "Bucket" },
            };
            threadBatchWrite.AddDocumentToPut(thread1);

            // Specify item to delete from the Thread table.
            threadBatchWrite.AddKeyToDelete("someForumName", "someSubject");

            // Create multi-table batch.
            var superBatch = new MultiTableDocumentBatchWrite();
            superBatch.AddBatch(forumBatchWrite);
            superBatch.AddBatch(threadBatchWrite);
            Console.WriteLine("Performing batch write in MultiTableBatchWrite()");

            // Execute the batch.
            await superBatch.ExecuteAsync();
        }
    }
```

ドキュメントモデルを使用してテーブルをスキャンします。

```
    /// <summary>
```

```
    /// Shows how to use mid-level Amazon DynamoDB API calls to scan a DynamoDB
    /// table for values.
    /// </summary>
    public class MidLevelScanOnly
    {
        public static async Task Main()
        {
            IAmazonDynamoDB client = new AmazonDynamoDBClient();

            Table productCatalogTable = Table.LoadTable(client, "ProductCatalog");

            await FindProductsWithNegativePrice(productCatalogTable);
            await FindProductsWithNegativePriceWithConfig(productCatalogTable);
        }

        /// <summary>
        /// Retrieves any products that have a negative price in a DynamoDB table.
        /// </summary>
        /// <param name="productCatalogTable">A DynamoDB table object.</param>
        public static async Task FindProductsWithNegativePrice(
          Table productCatalogTable)
        {
            // Assume there is a price error. So we scan to find items priced < 0.
            var scanFilter = new ScanFilter();
            scanFilter.AddCondition("Price", ScanOperator.LessThan, 0);

            Search search = productCatalogTable.Scan(scanFilter);

            do
            {
                var documentList = await search.GetNextSetAsync();
                Console.WriteLine("\nFindProductsWithNegativePrice:
printing .............");

                foreach (var document in documentList)
                {
                    PrintDocument(document);
                }
            }
            while (!search.IsDone);
        }

        /// <summary>
        /// Finds any items in the ProductCatalog table using a DynamoDB
```

```
        /// configuration object.
        /// </summary>
        /// <param name="productCatalogTable">A DynamoDB table object.</param>
        public static async Task FindProductsWithNegativePriceWithConfig(
          Table productCatalogTable)
        {
            // Assume there is a price error. So we scan to find items priced < 0.
            var scanFilter = new ScanFilter();
            scanFilter.AddCondition("Price", ScanOperator.LessThan, 0);

            var config = new ScanOperationConfig()
            {
                Filter = scanFilter,
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string> { "Title", "Id" },
            };

            Search search = productCatalogTable.Scan(config);

            do
            {
                var documentList = await search.GetNextSetAsync();
                Console.WriteLine("\nFindProductsWithNegativePriceWithConfig:
printing .....");

                foreach (var document in documentList)
                {
                    PrintDocument(document);
                }
            }
            while (!search.IsDone);
        }

        /// <summary>
        /// Displays the details of the passed DynamoDB document object on the
        /// console.
        /// </summary>
        /// <param name="document">A DynamoDB document object.</param>
        public static void PrintDocument(Document document)
        {
            Console.WriteLine();
            foreach (var attribute in document.GetAttributeNames())
            {
                string stringValue = null;
```

```
                var value = document[attribute];
                if (value is Primitive)
                {
                    stringValue = value.AsPrimitive().Value.ToString();
                }
                else if (value is PrimitiveList)
                {
                    stringValue = string.Join(",", (from primitive
                      in value.AsPrimitiveList().Entries
                                                    select
primitive.Value).ToArray());
                }

                Console.WriteLine($"{attribute} - {stringValue}");
            }
        }
    }
```

ドキュメントモデルを使用して、テーブルをクエリおよびスキャンする。

```
    /// <summary>
    /// Shows how to perform mid-level query procedures on an Amazon DynamoDB
    /// table.
    /// </summary>
    public class MidLevelQueryAndScan
    {
        public static async Task Main()
        {
            IAmazonDynamoDB client = new AmazonDynamoDBClient();

            // Query examples.
            Table replyTable = Table.LoadTable(client, "Reply");
            string forumName = "Amazon DynamoDB";
            string threadSubject = "DynamoDB Thread 2";

            await FindRepliesInLast15Days(replyTable);
            await FindRepliesInLast15DaysWithConfig(replyTable, forumName,
threadSubject);
            await FindRepliesPostedWithinTimePeriod(replyTable, forumName,
threadSubject);
```

```
            // Get Example.
            Table productCatalogTable = Table.LoadTable(client, "ProductCatalog");
            int productId = 101;

            await GetProduct(productCatalogTable, productId);
        }

        /// <summary>
        /// Retrieves information about a product from the DynamoDB table
        /// ProductCatalog based on the product ID and displays the information
        /// on the console.
        /// </summary>
        /// <param name="tableName">The name of the table from which to retrieve
        /// product information.</param>
        /// <param name="productId">The ID of the product to retrieve.</param>
        public static async Task GetProduct(Table tableName, int productId)
        {
            Console.WriteLine("*** Executing GetProduct() ***");
            Document productDocument = await tableName.GetItemAsync(productId);
            if (productDocument != null)
            {
                PrintDocument(productDocument);
            }
            else
            {
                Console.WriteLine("Error: product " + productId + " does not
exist");
            }
        }

        /// <summary>
        /// Retrieves replies from the passed DynamoDB table object.
        /// </summary>
        /// <param name="table">The table we want to query.</param>
        public static async Task FindRepliesInLast15Days(
          Table table)
        {
            DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
            var filter = new QueryFilter("Id", QueryOperator.Equal, "Id");
            filter.AddCondition("ReplyDateTime", QueryOperator.GreaterThan,
twoWeeksAgoDate);

            // Use Query overloads that take the minimum required query parameters.
```

```
            Search search = table.Query(filter);

            do
            {
                var documentSet = await search.GetNextSetAsync();
                Console.WriteLine("\nFindRepliesInLast15Days:
printing ............");

                foreach (var document in documentSet)
                {
                    PrintDocument(document);
                }
            }
            while (!search.IsDone);
        }

        /// <summary>
        /// Retrieve replies made during a specific time period.
        /// </summary>
        /// <param name="table">The table we want to query.</param>
        /// <param name="forumName">The name of the forum that we're interested
in.</param>
        /// <param name="threadSubject">The subject of the thread, which we are
        /// searching for replies.</param>
        public static async Task FindRepliesPostedWithinTimePeriod(
          Table table,
          string forumName,
          string threadSubject)
        {
            DateTime startDate = DateTime.UtcNow.Subtract(new TimeSpan(21, 0, 0,
0));
            DateTime endDate = DateTime.UtcNow.Subtract(new TimeSpan(1, 0, 0, 0));

            var filter = new QueryFilter("Id", QueryOperator.Equal, forumName + "#"
+ threadSubject);
            filter.AddCondition("ReplyDateTime", QueryOperator.Between, startDate,
endDate);

            var config = new QueryOperationConfig()
            {
                Limit = 2, // 2 items/page.
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string>
        {
```

```
          "Message",
          "ReplyDateTime",
          "PostedBy",
        },
                ConsistentRead = true,
                Filter = filter,
            };

            Search search = table.Query(config);

            do
            {
                var documentList = await search.GetNextSetAsync();
                Console.WriteLine("\nFindRepliesPostedWithinTimePeriod: printing
replies posted within dates: {0} and {1} .....", startDate, endDate);

                foreach (var document in documentList)
                {
                    PrintDocument(document);
                }
            }
            while (!search.IsDone);
        }

        /// <summary>
        /// Perform a query for replies made in the last 15 days using a DynamoDB
        /// QueryOperationConfig object.
        /// </summary>
        /// <param name="table">The table we want to query.</param>
        /// <param name="forumName">The name of the forum that we're interested
in.</param>
        /// <param name="threadName">The bane of the thread that we are searching
        /// for replies.</param>
        public static async Task FindRepliesInLast15DaysWithConfig(
          Table table,
          string forumName,
          string threadName)
        {
            DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
            var filter = new QueryFilter("Id", QueryOperator.Equal, forumName + "#"
+ threadName);
            filter.AddCondition("ReplyDateTime", QueryOperator.GreaterThan,
twoWeeksAgoDate);
```

```
            var config = new QueryOperationConfig()
            {
                Filter = filter,

                // Optional parameters.
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string>
                {
                  "Message",
                  "ReplyDateTime",
                  "PostedBy",
                },
                ConsistentRead = true,
            };

            Search search = table.Query(config);

            do
            {
                var documentSet = await search.GetNextSetAsync();
                Console.WriteLine("\nFindRepliesInLast15DaysWithConfig:
printing ............");

                foreach (var document in documentSet)
                {
                    PrintDocument(document);
                }
            }
            while (!search.IsDone);
        }

        /// <summary>
        /// Displays the contents of the passed DynamoDB document on the console.
        /// </summary>
        /// <param name="document">A DynamoDB document to display.</param>
        public static void PrintDocument(Document document)
        {
            Console.WriteLine();
            foreach (var attribute in document.GetAttributeNames())
            {
                string stringValue = null;
                var value = document[attribute];

                if (value is Primitive)
```

```
                {
                    stringValue = value.AsPrimitive().Value.ToString();
                }
                else if (value is PrimitiveList)
                {
                    stringValue = string.Join(",", (from primitive
                      in value.AsPrimitiveList().Entries
                                                    select
primitive.Value).ToArray());
                }

                Console.WriteLine($"{attribute} - {stringValue}");
            }
        }
    }
```

高レベルのオブジェクト永続性モデルを使用する

次のコード例は、DynamoDB と AWS SDK のオブジェクト永続性モデルを使用して、作成、読み取り、更新、削除 (CRUD) およびバッチオペレーションを実行する方法を示しています。

詳細については、「オブジェクト永続性モデル」を参照してください。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

高レベルのオブジェクト永続性モデルを使用して CRUD オペレーションを実行します。

```
    /// <summary>
    /// Shows how to perform high-level CRUD operations on an Amazon DynamoDB
    /// table.
    /// </summary>
    public class HighLevelItemCrud
    {
```

```
        public static async Task Main()
        {
            var client = new AmazonDynamoDBClient();
            DynamoDBContext context = new DynamoDBContext(client);
            await PerformCRUDOperations(context);
        }

        public static async Task PerformCRUDOperations(IDynamoDBContext context)
        {
            int bookId = 1001; // Some unique value.
            Book myBook = new Book
            {
                Id = bookId,
                Title = "object persistence-AWS SDK for.NET SDK-Book 1001",
                Isbn = "111-1111111001",
                BookAuthors = new List<string> { "Author 1", "Author 2" },
            };

            // Save the book to the ProductCatalog table.
            await context.SaveAsync(myBook);

            // Retrieve the book from the ProductCatalog table.
            Book bookRetrieved = await context.LoadAsync<Book>(bookId);

            // Update some properties.
            bookRetrieved.Isbn = "222-2222221001";

            // Update existing authors list with the following values.
            bookRetrieved.BookAuthors = new List<string> { " Author 1", "Author
x" };
            await context.SaveAsync(bookRetrieved);

            // Retrieve the updated book. This time, add the optional
            // ConsistentRead parameter using DynamoDBContextConfig object.
            await context.LoadAsync<Book>(bookId, new DynamoDBContextConfig
            {
                ConsistentRead = true,
            });

            // Delete the book.
            await context.DeleteAsync<Book>(bookId);

            // Try to retrieve deleted book. It should return null.
```

```
            Book deletedBook = await context.LoadAsync<Book>(bookId, new
DynamoDBContextConfig
            {
                ConsistentRead = true,
            });

            if (deletedBook == null)
            {
                Console.WriteLine("Book is deleted");
            }
        }
    }
```

高レベルのオブジェクト永続性モデルを使用してバッチ書き込みオペレーションを実行します。

```
    /// <summary>
    /// Performs high-level batch write operations to an Amazon DynamoDB table.
    /// This example was written using the AWS SDK for .NET version 3.7 and .NET
    /// Core 5.0.
    /// </summary>
    public class HighLevelBatchWriteItem
    {
        public static async Task SingleTableBatchWrite(IDynamoDBContext context)
        {
            Book book1 = new Book
            {
                Id = 902,
                InPublication = true,
                Isbn = "902-11-11-1111",
                PageCount = "100",
                Price = 10,
                ProductCategory = "Book",
                Title = "My book3 in batch write",
            };

            Book book2 = new Book
            {
                Id = 903,
                InPublication = true,
                Isbn = "903-11-11-1111",
```

```
                PageCount = "200",
                Price = 10,
                ProductCategory = "Book",
                Title = "My book4 in batch write",
            };

            var bookBatch = context.CreateBatchWrite<Book>();
            bookBatch.AddPutItems(new List<Book> { book1, book2 });

            Console.WriteLine("Adding two books to ProductCatalog table.");
            await bookBatch.ExecuteAsync();
        }

        public static async Task MultiTableBatchWrite(IDynamoDBContext context)
        {
            // New Forum item.
            Forum newForum = new Forum
            {
                Name = "Test BatchWrite Forum",
                Threads = 0,
            };
            var forumBatch = context.CreateBatchWrite<Forum>();
            forumBatch.AddPutItem(newForum);

            // New Thread item.
            Thread newThread = new Thread
            {
                ForumName = "S3 forum",
                Subject = "My sample question",
                KeywordTags = new List<string> { "S3", "Bucket" },
                Message = "Message text",
            };

            DynamoDBOperationConfig config = new DynamoDBOperationConfig();
            config.SkipVersionCheck = true;
            var threadBatch = context.CreateBatchWrite<Thread>(config);
            threadBatch.AddPutItem(newThread);
            threadBatch.AddDeleteKey("some partition key value", "some sort key
value");

            var superBatch = new MultiTableBatchWrite(forumBatch, threadBatch);

            Console.WriteLine("Performing batch write in MultiTableBatchWrite().");
            await superBatch.ExecuteAsync();
```

```
        }

        public static async Task Main()
        {
            AmazonDynamoDBClient client = new AmazonDynamoDBClient();
            DynamoDBContext context = new DynamoDBContext(client);

            await SingleTableBatchWrite(context);
            await MultiTableBatchWrite(context);
        }
    }
```

オブジェクト永続性モデルを使用して、任意のデータをテーブルにマッピングします。

```
    /// <summary>
    /// Shows how to map arbitrary data to an Amazon DynamoDB table.
    /// </summary>
    public class HighLevelMappingArbitraryData
    {
        /// <summary>
        /// Creates a book, adds it to the DynamoDB ProductCatalog table, retrieves
        /// the new book from the table, updates the dimensions and writes the
        /// changed item back to the table.
        /// </summary>
        /// <param name="context">The DynamoDB context object used to write and
        /// read data from the table.</param>
        public static async Task AddRetrieveUpdateBook(IDynamoDBContext context)
        {
            // Create a book.
            DimensionType myBookDimensions = new DimensionType()
            {
                Length = 8M,
                Height = 11M,
                Thickness = 0.5M,
            };

            Book myBook = new Book
            {
                Id = 501,
```

```
                Title = "AWS SDK for .NET Object Persistence Model Handling
Arbitrary Data",
                Isbn = "999-9999999999",
                BookAuthors = new List<string> { "Author 1", "Author 2" },
                Dimensions = myBookDimensions,
            };

            // Add the book to the DynamoDB table ProductCatalog.
            await context.SaveAsync(myBook);

            // Retrieve the book.
            Book bookRetrieved = await context.LoadAsync<Book>(501);

            // Update the book dimensions property.
            bookRetrieved.Dimensions.Height += 1;
            bookRetrieved.Dimensions.Length += 1;
            bookRetrieved.Dimensions.Thickness += 0.2M;

            // Write the changed item to the table.
            await context.SaveAsync(bookRetrieved);
        }

        public static async Task Main()
        {
            var client = new AmazonDynamoDBClient();
            DynamoDBContext context = new DynamoDBContext(client);
            await AddRetrieveUpdateBook(context);
        }
    }
```

高レベルのオブジェクト永続性モデルを使用してテーブルをクエリおよびスキャンします。

```
    /// <summary>
    /// Shows how to perform high-level query and scan operations to Amazon
    /// DynamoDB tables.
    /// </summary>
    public class HighLevelQueryAndScan
    {
        public static async Task Main()
        {
```

```
            var client = new AmazonDynamoDBClient();

            DynamoDBContext context = new DynamoDBContext(client);

            // Get an item.
            await GetBook(context, 101);

            // Sample forum and thread to test queries.
            string forumName = "Amazon DynamoDB";
            string threadSubject = "DynamoDB Thread 1";

            // Sample queries.
            await FindRepliesInLast15Days(context, forumName, threadSubject);
            await FindRepliesPostedWithinTimePeriod(context, forumName,
 threadSubject);

            // Scan table.
            await FindProductsPricedLessThanZero(context);
        }

        public static async Task GetBook(IDynamoDBContext context, int productId)
        {
            Book bookItem = await context.LoadAsync<Book>(productId);

            Console.WriteLine("\nGetBook: Printing result.....");
            Console.WriteLine($"Title: {bookItem.Title} \n ISBN:{bookItem.Isbn} \n
 No. of pages: {bookItem.PageCount}");
        }

        /// <summary>
        /// Queries a DynamoDB table to find replies posted within the last 15 days.
        /// </summary>
        /// <param name="context">The DynamoDB context used to perform the query.</
param>
        /// <param name="forumName">The name of the forum that we're interested
 in.</param>
        /// <param name="threadSubject">The thread object containing the query
 parameters.</param>
        public static async Task FindRepliesInLast15Days(
          IDynamoDBContext context,
          string forumName,
          string threadSubject)
        {
            string replyId = $"{forumName} #{threadSubject}";
```

```
            DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);

            List<object> times = new List<object>();
            times.Add(twoWeeksAgoDate);

            List<ScanCondition> scs = new List<ScanCondition>();
            var sc = new ScanCondition("PostedBy", ScanOperator.GreaterThan,
 times.ToArray());
            scs.Add(sc);

            var cfg = new DynamoDBOperationConfig
            {
                QueryFilter = scs,
            };

            AsyncSearch<Reply> response = context.QueryAsync<Reply>(replyId, cfg);
            IEnumerable<Reply> latestReplies = await response.GetRemainingAsync();

            Console.WriteLine("\nReplies in last 15 days:");

            foreach (Reply r in latestReplies)
            {

 Console.WriteLine($"{r.Id}\t{r.PostedBy}\t{r.Message}\t{r.ReplyDateTime}");
            }
        }

        /// <summary>
        /// Queries for replies posted within a specific time period.
        /// </summary>
        /// <param name="context">The DynamoDB context used to perform the query.</
param>
        /// <param name="forumName">The name of the forum that we're interested
 in.</param>
        /// <param name="threadSubject">Information about the subject that we're
        /// interested in.</param>
        public static async Task FindRepliesPostedWithinTimePeriod(
          IDynamoDBContext context,
          string forumName,
          string threadSubject)
        {
            string forumId = forumName + "#" + threadSubject;
            Console.WriteLine("\nReplies posted within time period:");
```

```
            DateTime startDate = DateTime.UtcNow - TimeSpan.FromDays(30);
            DateTime endDate = DateTime.UtcNow - TimeSpan.FromDays(1);

            List<object> times = new List<object>();
            times.Add(startDate);
            times.Add(endDate);

            List<ScanCondition> scs = new List<ScanCondition>();
            var sc = new ScanCondition("LastPostedBy", ScanOperator.Between,
times.ToArray());
            scs.Add(sc);

            var cfg = new DynamoDBOperationConfig
            {
                QueryFilter = scs,
            };

            AsyncSearch<Reply> response = context.QueryAsync<Reply>(forumId, cfg);
            IEnumerable<Reply> repliesInAPeriod = await
response.GetRemainingAsync();

            foreach (Reply r in repliesInAPeriod)
            {

Console.WriteLine("{r.Id}\t{r.PostedBy}\t{r.Message}\t{r.ReplyDateTime}");
            }
        }

        /// <summary>
        /// Queries the DynamoDB ProductCatalog table for products costing less
        /// than zero.
        /// </summary>
        /// <param name="context">The DynamoDB context object used to perform the
        /// query.</param>
        public static async Task FindProductsPricedLessThanZero(IDynamoDBContext
context)
        {
            int price = 0;

            List<ScanCondition> scs = new List<ScanCondition>();
            var sc1 = new ScanCondition("Price", ScanOperator.LessThan, price);
            var sc2 = new ScanCondition("ProductCategory", ScanOperator.Equal,
"Book");
            scs.Add(sc1);
```

```
            scs.Add(sc2);

            AsyncSearch<Book> response = context.ScanAsync<Book>(scs);

            IEnumerable<Book> itemsWithWrongPrice = await
response.GetRemainingAsync();

            Console.WriteLine("\nFindProductsPricedLessThanZero: Printing
result.....");

            foreach (Book r in itemsWithWrongPrice)
            {
                Console.WriteLine($"{r.Id}\t{r.Title}\t{r.Price}\t{r.Isbn}");
            }
        }
    }
```

サーバーレスサンプル

DynamoDB トリガーから Lambda 関数を呼び出す

次のコード例は、DynamoDB ストリームからレコードを受信することによってトリガーされるイベントを受信する Lambda 関数を実装する方法を示しています。関数は DynamoDB ペイロードを取得し、レコードの内容をログ記録します。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

.NET を使用して Lambda で DynamoDB イベントの消費。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
```

```
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace AWSLambda_DDB;

public class Function
{
    public void FunctionHandler(DynamoDBEvent dynamoEvent, ILambdaContext context)
    {
        context.Logger.LogInformation($"Beginning to process
 {dynamoEvent.Records.Count} records...");

        foreach (var record in dynamoEvent.Records)
        {
            context.Logger.LogInformation($"Event ID: {record.EventID}");
            context.Logger.LogInformation($"Event Name: {record.EventName}");

            context.Logger.LogInformation(JsonSerializer.Serialize(record));
        }

        context.Logger.LogInformation("Stream processing complete.");
    }
}
```

DynamoDB トリガーで Lambda 関数のバッチアイテムの失敗をレポートする

次のコード例は、DynamoDB ストリームからイベントを受信する Lambda 関数に部分的なバッチレスポンスを実装する方法を示しています。この関数は、レスポンスとしてバッチアイテムの失敗を報告し、対象のメッセージを後で再試行するよう Lambda に伝えます。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

.NET を使用して Lambda で DynamoDB のバッチアイテム失敗のレポート。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace AWSLambda_DDB;

public class Function
{
    public StreamsEventResponse FunctionHandler(DynamoDBEvent dynamoEvent,
 ILambdaContext context)

    {
        context.Logger.LogInformation($"Beginning to process
 {dynamoEvent.Records.Count} records...");
        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
 List<StreamsEventResponse.BatchItemFailure>();
        StreamsEventResponse streamsEventResponse = new StreamsEventResponse();

        foreach (var record in dynamoEvent.Records)
        {
            try
            {
                var sequenceNumber = record.Dynamodb.SequenceNumber;
                context.Logger.LogInformation(sequenceNumber);
            }
```

```
            catch (Exception ex)
            {
                context.Logger.LogError(ex.Message);
                batchItemFailures.Add(new StreamsEventResponse.BatchItemFailure()
 { ItemIdentifier = record.Dynamodb.SequenceNumber });
            }
        }

        if (batchItemFailures.Count > 0)
        {
            streamsEventResponse.BatchItemFailures = batchItemFailures;
        }

        context.Logger.LogInformation("Stream processing complete.");
        return streamsEventResponse;
    }
}
```

を使用した Amazon EC2 の例 AWS SDK for .NET

次のコード例は、Amazon EC2 AWS SDK for .NET で 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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。このリンクには GitHub、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が記載されています。

開始方法

Hello Amazon EC2

以下のコード例は、Amazon EC2 の利用開始方法を表示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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>A Task object.</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>();

        var request = new DescribeSecurityGroupsRequest
        {
            MaxResults = 10,
        };


        // Retrieve information about up to 10 Amazon EC2 security groups.
        var response = await ec2Client.DescribeSecurityGroupsAsync(request);
```

```
        // Now print the security groups returned by the call to
        // DescribeSecurityGroupsAsync.
        Console.WriteLine("Security Groups:");
        response.SecurityGroups.ForEach(group =>
        {
            Console.WriteLine($"Security group: {group.GroupName} ID:
 {group.GroupId}");
        });
    }
}
```

- API の詳細については、「 API リファレンスDescribeSecurityGroups」の「」を参照してください。 AWS SDK for .NET

トピック

- アクション
- シナリオ

アクション

`AllocateAddress`

次の例は、`AllocateAddress` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Allocate an Elastic IP address.
    /// </summary>
    /// <returns>The allocation Id of the allocated address.</returns>
    public async Task<string> AllocateAddress()
    {
        var request = new AllocateAddressRequest();
```

```
        var response = await _amazonEC2.AllocateAddressAsync(request);
        return response.AllocationId;
    }
```

- API の詳細については、「 API リファレンスAllocateAddress」の「」を参照してください。 AWS SDK for .NET

`AssociateAddress`

次の例は、`AssociateAddress` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Associate an Elastic IP address to an EC2 instance.
    /// </summary>
    /// <param name="allocationId">The allocation Id of an Elastic IP address.</
param>
    /// <param name="instanceId">The instance Id of the EC2 instance to
    /// associate the address with.</param>
    /// <returns>The association Id that represents
    /// the association of the Elastic IP address with an instance.</returns>
    public async Task<string> AssociateAddress(string allocationId, string
 instanceId)
    {
        var request = new AssociateAddressRequest
        {
            AllocationId = allocationId,
            InstanceId = instanceId
        };

        var response = await _amazonEC2.AssociateAddressAsync(request);
        return response.AssociationId;
```

```
    }
```

- API の詳細については、「 API リファレンスAssociateAddress」の「」を参照してください。 AWS SDK for .NET

`AuthorizeSecurityGroupIngress`

次の例は、`AuthorizeSecurityGroupIngress` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Authorize the local computer ingress to EC2 instances associated
    /// with the virtual private cloud (VPC) security group.
    /// </summary>
    /// <param name="groupName">The name of the security group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AuthorizeSecurityGroupIngress(string groupName)
    {
        // Get the IP address for the local computer.
        var ipAddress = await GetIpAddress();
        Console.WriteLine($"Your IP address is: {ipAddress}");
        var ipRanges = new List<IpRange> { new IpRange { CidrIp =
$"{ipAddress}/32" } };
        var permission = new IpPermission
        {
            Ipv4Ranges = ipRanges,
            IpProtocol = "tcp",
            FromPort = 22,
            ToPort = 22
        };
        var permissions = new List<IpPermission> { permission };
        var response = await _amazonEC2.AuthorizeSecurityGroupIngressAsync(
            new AuthorizeSecurityGroupIngressRequest(groupName, permissions));
```

```
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Authorize the local computer for ingress to
    /// the Amazon EC2 SecurityGroup.
    /// </summary>
    /// <returns>The IPv4 address of the computer running the scenario.</returns>
    private static async Task<string> GetIpAddress()
    {
        var httpClient = new HttpClient();
        var ipString = await httpClient.GetStringAsync("https://
checkip.amazonaws.com");

        // The IP address is returned with a new line
        // character on the end. Trim off the whitespace and
        // return the value to the caller.
        return ipString.Trim();
    }
```

- API の詳細については、「 API リファレンスAuthorizeSecurityGroupIngress」の「」を参照してください。 AWS SDK for .NET

`CreateKeyPair`

次の例は、`CreateKeyPair` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Create an Amazon EC2 key pair.
    /// </summary>
    /// <param name="keyPairName">The name for the new key pair.</param>
    /// <returns>The Amazon EC2 key pair created.</returns>
```

```
    public async Task<KeyPair?> CreateKeyPair(string keyPairName)
    {
        var request = new CreateKeyPairRequest
        {
            KeyName = keyPairName,
        };

        var response = await _amazonEC2.CreateKeyPairAsync(request);

        if (response.HttpStatusCode == HttpStatusCode.OK)
        {
            var kp = response.KeyPair;
            return kp;
        }
        else
        {
            Console.WriteLine("Could not create key pair.");
            return null;
        }
    }

    /// <summary>
    /// Save KeyPair information to a temporary file.
    /// </summary>
    /// <param name="keyPair">The name of the key pair.</param>
    /// <returns>The full path to the temporary file.</returns>
    public string SaveKeyPair(KeyPair keyPair)
    {
        var tempPath = Path.GetTempPath();
        var tempFileName = $"{tempPath}\\{Path.GetRandomFileName()}";
        var pemFileName = Path.ChangeExtension(tempFileName, "pem");

        // Save the key pair to a file in a temporary folder.
        using var stream = new FileStream(pemFileName, FileMode.Create);
        using var writer = new StreamWriter(stream);
        writer.WriteLine(keyPair.KeyMaterial);

        return pemFileName;
    }
```

- API の詳細については、「 API リファレンスCreateKeyPair」の「」を参照してください。AWS SDK for .NET

CreateLaunchTemplate

次の例は、CreateLaunchTemplate を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
   /// <summary>
   /// Creates an Amazon EC2 launch template to use with Amazon EC2 Auto Scaling.
   /// The launch template specifies a Bash script in its user data field that runs
after
   /// the instance is started. This script installs the Python packages and starts
a Python
   /// web server on the instance.
   /// </summary>
   /// <param name="startupScriptPath">The path to a Bash script file that is
run.</param>
   /// <param name="instancePolicyPath">The path to a permissions policy to create
and attach to the profile.</param>
   /// <returns>The template object.</returns>
   public async Task<Amazon.EC2.Model.LaunchTemplate> CreateTemplate(string
startupScriptPath, string instancePolicyPath)
   {
       await CreateKeyPair(_keyPairName);
       await CreateInstanceProfileWithName(_instancePolicyName, _instanceRoleName,
_instanceProfileName, instancePolicyPath);

       var startServerText = await File.ReadAllTextAsync(startupScriptPath);
       var plainTextBytes = System.Text.Encoding.UTF8.GetBytes(startServerText);

       var amiLatest = await _amazonSsm.GetParameterAsync(
           new GetParameterRequest() { Name = _amiParam });
       var amiId = amiLatest.Parameter.Value;
       var launchTemplateResponse = await _amazonEc2.CreateLaunchTemplateAsync(
           new CreateLaunchTemplateRequest()
           {
               LaunchTemplateName = _launchTemplateName,
               LaunchTemplateData = new RequestLaunchTemplateData()
```

```
                {
                    InstanceType = _instanceType,
                    ImageId = amiId,
                    IamInstanceProfile =
                        new
                            LaunchTemplateIamInstanceProfileSpecificationRequest()
                        {
                            Name = _instanceProfileName
                        },
                    KeyName = _keyPairName,
                    UserData = System.Convert.ToBase64String(plainTextBytes)
                }
            });
        return launchTemplateResponse.LaunchTemplate;

    }
```

- API の詳細については、「 API リファレンスCreateLaunchTemplate」の「」を参照してください。 AWS SDK for .NET

`CreateSecurityGroup`

次の例は、`CreateSecurityGroup` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Create an Amazon EC2 security group.
    /// </summary>
    /// <param name="groupName">The name for the new security group.</param>
    /// <param name="groupDescription">A description of the new security group.</
param>
    /// <returns>The group Id of the new security group.</returns>
```

```
    public async Task<string> CreateSecurityGroup(string groupName, string
groupDescription)
    {
        var response = await _amazonEC2.CreateSecurityGroupAsync(
            new CreateSecurityGroupRequest(groupName, groupDescription));

        return response.GroupId;
    }
```

- API の詳細については、「 API リファレンスCreateSecurityGroup」の「」を参照してください。 AWS SDK for .NET

`DeleteKeyPair`

次の例は、DeleteKeyPair を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete an Amazon EC2 key pair.
    /// </summary>
    /// <param name="keyPairName">The name of the key pair to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteKeyPair(string keyPairName)
    {
        try
        {
            await _amazonEC2.DeleteKeyPairAsync(new
DeleteKeyPairRequest(keyPairName)).ConfigureAwait(false);
            return true;
        }
        catch (Exception ex)
        {
```

```
            Console.WriteLine($"Couldn't delete the key pair because:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Delete the temporary file where the key pair information was saved.
    /// </summary>
    /// <param name="tempFileName">The path to the temporary file.</param>
    public void DeleteTempFile(string tempFileName)
    {
        if (File.Exists(tempFileName))
        {
            File.Delete(tempFileName);
        }
    }
```

- API の詳細については、「 API リファレンスDeleteKeyPair」の「」を参照してください。AWS SDK for .NET

`DeleteLaunchTemplate`

次の例は、`DeleteLaunchTemplate` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete a launch template by name.
    /// </summary>
    /// <param name="templateName">The name of the template to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteTemplateByName(string templateName)
    {
```

```
        try
        {
            await _amazonEc2.DeleteLaunchTemplateAsync(
                new DeleteLaunchTemplateRequest()
                {
                    LaunchTemplateName = templateName
                });
        }
        catch (AmazonClientException)
        {
            Console.WriteLine($"Unable to delete template {templateName}.");
        }
    }
```

- API の詳細については、「 API リファレンスDeleteLaunchTemplate」の「」を参照してください。 AWS SDK for .NET

`DeleteSecurityGroup`

次の例は、DeleteSecurityGroup を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete an Amazon EC2 security group.
    /// </summary>
    /// <param name="groupName">The name of the group to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteSecurityGroup(string groupId)
    {
        var response = await _amazonEC2.DeleteSecurityGroupAsync(new
DeleteSecurityGroupRequest { GroupId = groupId });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスDeleteSecurityGroup」の「」を参照してください。 AWS SDK for .NET

`DescribeAvailabilityZones`

次の例は、`DescribeAvailabilityZones` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
/// Get a list of Availability Zones in the AWS Region of the Amazon EC2 Client.
/// </summary>
/// <returns>A list of availability zones.</returns>
public async Task<List<string>> DescribeAvailabilityZones()
{
    var zoneResponse = await _amazonEc2.DescribeAvailabilityZonesAsync(
        new DescribeAvailabilityZonesRequest());
    return zoneResponse.AvailabilityZones.Select(z => z.ZoneName).ToList();
}
```

- API の詳細については、「 API リファレンスDescribeAvailabilityZones」の「」を参照してください。 AWS SDK for .NET

`DescribeIamInstanceProfileAssociations`

次の例は、`DescribeIamInstanceProfileAssociations` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get the instance profile association data for an instance.
    /// </summary>
    /// <param name="instanceId">The Id of the instance.</param>
    /// <returns>Instance profile associations data.</returns>
    public async Task<IamInstanceProfileAssociation> GetInstanceProfile(string
instanceId)
    {
        var response = await _amazonEc2.DescribeIamInstanceProfileAssociationsAsync(
            new DescribeIamInstanceProfileAssociationsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("instance-id", new List<string>() { instanceId })
                },
            });
        return response.IamInstanceProfileAssociations[0];
    }
```

- API の詳細については、「 API リファレンスDescribeIamInstanceProfileAssociations」の「」を参照してください。 AWS SDK for .NET

`DescribeInstanceTypes`

次の例は、DescribeInstanceTypes を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Describe the instance types available.
    /// </summary>
    /// <returns>A list of instance type information.</returns>
    public async Task<List<InstanceTypeInfo>>
DescribeInstanceTypes(ArchitectureValues architecture)
    {
        var request = new DescribeInstanceTypesRequest();

        var filters = new List<Filter>
            { new Filter("processor-info.supported-architecture", new List<string>
{ architecture.ToString() }) };
        filters.Add(new Filter("instance-type", new() { "*.micro", "*.small" }));

        request.Filters = filters;
        var instanceTypes = new List<InstanceTypeInfo>();

        var paginator = _amazonEC2.Paginators.DescribeInstanceTypes(request);
        await foreach (var instanceType in paginator.InstanceTypes)
        {
            instanceTypes.Add(instanceType);
        }
        return instanceTypes;
    }
```

- API の詳細については、「 API リファレンスDescribeInstanceTypes」の「」を参照してください。 AWS SDK for .NET

`DescribeInstances`

次の例は、DescribeInstances を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get information about existing EC2 images.
    /// </summary>
    /// <returns>Async task.</returns>
    public async Task DescribeInstances()
    {
        // List all EC2 instances.
        await GetInstanceDescriptions();

        string tagName = "IncludeInList";
        string tagValue = "Yes";
        await GetInstanceDescriptionsFiltered(tagName, tagValue);
    }

    /// <summary>
    /// Get information for all existing Amazon EC2 instances.
    /// </summary>
    /// <returns>Async task.</returns>
    public async Task GetInstanceDescriptions()
    {
        Console.WriteLine("Showing all instances:");
        var paginator = _amazonEC2.Paginators.DescribeInstances(new
DescribeInstancesRequest());

        await foreach (var response in paginator.Responses)
        {
            foreach (var reservation in response.Reservations)
            {
                foreach (var instance in reservation.Instances)
                {
                    Console.Write($"Instance ID: {instance.InstanceId}");
                    Console.WriteLine($"\tCurrent State: {instance.State.Name}");
                }
            }
```

```
        }
    }

    /// <summary>
    /// Get information about EC2 instances filtered by a tag name and value.
    /// </summary>
    /// <param name="tagName">The name of the tag to filter on.</param>
    /// <param name="tagValue">The value of the tag to look for.</param>
    /// <returns>Async task.</returns>
    public async Task GetInstanceDescriptionsFiltered(string tagName, string
tagValue)
    {
        // This tag filters the results of the instance list.
        var filters = new List<Filter>
        {
            new Filter
            {
                Name = $"tag:{tagName}",
                Values = new List<string>
                {
                    tagValue,
                },
            },
        };
        var request = new DescribeInstancesRequest
        {
            Filters = filters,
        };

        Console.WriteLine("\nShowing instances with tag: \"IncludeInList\" set to
\"Yes\".");
        var paginator = _amazonEC2.Paginators.DescribeInstances(request);

        await foreach (var response in paginator.Responses)
        {
            foreach (var reservation in response.Reservations)
            {
                foreach (var instance in reservation.Instances)
                {
                    Console.Write($"Instance ID: {instance.InstanceId} ");
                    Console.WriteLine($"\tCurrent State: {instance.State.Name}");
                }
            }
        }
```

```
    }
```

- API の詳細については、「 API リファレンスDescribeInstances」の「」を参照してください。AWS SDK for .NET

`DescribeKeyPairs`

次の例は、`DescribeKeyPairs` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
/// Get information about an Amazon EC2 key pair.
/// </summary>
/// <param name="keyPairName">The name of the key pair.</param>
/// <returns>A list of key pair information.</returns>
public async Task<List<KeyPairInfo>> DescribeKeyPairs(string keyPairName)
{
    var request = new DescribeKeyPairsRequest();
    if (!string.IsNullOrEmpty(keyPairName))
    {
        request = new DescribeKeyPairsRequest
        {
            KeyNames = new List<string> { keyPairName }
        };
    }
    var response = await _amazonEC2.DescribeKeyPairsAsync(request);
    return response.KeyPairs.ToList();
}
```

- API の詳細については、「 API リファレンスDescribeKeyPairs」の「」を参照してください。AWS SDK for .NET

DescribeSecurityGroups

次の例は、DescribeSecurityGroups を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
/// Retrieve information for an Amazon EC2 security group.
/// </summary>
/// <param name="groupId">The Id of the Amazon EC2 security group.</param>
/// <returns>A list of security group information.</returns>
public async Task<List<SecurityGroup>> DescribeSecurityGroups(string groupId)
{
    var request = new DescribeSecurityGroupsRequest();
    var groupIds = new List<string> { groupId };
    request.GroupIds = groupIds;

    var response = await _amazonEC2.DescribeSecurityGroupsAsync(request);
    return response.SecurityGroups;
}

/// <summary>
/// Display the information returned by the call to
/// DescribeSecurityGroupsAsync.
/// </summary>
/// <param name="securityGroup">A list of security group information.</param>
public void DisplaySecurityGroupInfoAsync(SecurityGroup securityGroup)
{
    Console.WriteLine($"{securityGroup.GroupName}");
    Console.WriteLine("Ingress permissions:");
    securityGroup.IpPermissions.ForEach(permission =>
    {
        Console.WriteLine($"\tFromPort: {permission.FromPort}");
        Console.WriteLine($"\tIpProtocol: {permission.IpProtocol}");

        Console.Write($"\tIpv4Ranges: ");
```

```
            permission.Ipv4Ranges.ForEach(range => { Console.Write($"{range.CidrIp}
"); });

            Console.WriteLine($"\n\tIpv6Ranges:");
            permission.Ipv6Ranges.ForEach(range =>
{ Console.Write($"{range.CidrIpv6} "); });

            Console.Write($"\n\tPrefixListIds: ");
            permission.PrefixListIds.ForEach(id => Console.Write($"{id.Id} "));

            Console.WriteLine($"\n\tTo Port: {permission.ToPort}");
        });
        Console.WriteLine("Egress permissions:");
        securityGroup.IpPermissionsEgress.ForEach(permission =>
        {
            Console.WriteLine($"\tFromPort: {permission.FromPort}");
            Console.WriteLine($"\tIpProtocol: {permission.IpProtocol}");

            Console.Write($"\tIpv4Ranges: ");
            permission.Ipv4Ranges.ForEach(range => { Console.Write($"{range.CidrIp}
"); });

            Console.WriteLine($"\n\tIpv6Ranges:");
            permission.Ipv6Ranges.ForEach(range =>
{ Console.Write($"{range.CidrIpv6} "); });

            Console.Write($"\n\tPrefixListIds: ");
            permission.PrefixListIds.ForEach(id => Console.Write($"{id.Id} "));

            Console.WriteLine($"\n\tTo Port: {permission.ToPort}");
        });
    }
```

- API の詳細については、「 API リファレンスDescribeSecurityGroups」の「」を参照してください。 AWS SDK for .NET

`DescribeSubnets`

次の例は、`DescribeSubnets` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get all the subnets for a Vpc in a set of availability zones.
    /// </summary>
    /// <param name="vpcId">The Id of the Vpc.</param>
    /// <param name="availabilityZones">The list of availability zones.</param>
    /// <returns>The collection of subnet objects.</returns>
    public async Task<List<Subnet>> GetAllVpcSubnetsForZones(string vpcId,
List<string> availabilityZones)
    {
        var subnets = new List<Subnet>();
        var subnetPaginator = _amazonEc2.Paginators.DescribeSubnets(
            new DescribeSubnetsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("vpc-id", new List<string>() { vpcId}),
                    new ("availability-zone", availabilityZones),
                    new ("default-for-az", new List<string>() { "true" })
                }
            });

        // Get the entire list using the paginator.
        await foreach (var subnet in subnetPaginator.Subnets)
        {
            subnets.Add(subnet);
        }

        return subnets;
    }
```

- API の詳細については、「 API リファレンスDescribeSubnets」の「」を参照してください。AWS SDK for .NET

DescribeVpcs

次の例は、DescribeVpcs を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
/// Get the default VPC for the account.
/// </summary>
/// <returns>The default VPC object.</returns>
public async Task<Vpc> GetDefaultVpc()
{
    var vpcResponse = await _amazonEc2.DescribeVpcsAsync(
        new DescribeVpcsRequest()
        {
            Filters = new List<Amazon.EC2.Model.Filter>()
            {
                new ("is-default", new List<string>() { "true" })
            }
        });
    return vpcResponse.Vpcs[0];
}
```

- API の詳細については、「 API リファレンスDescribeVpcs」の「」を参照してください。AWS SDK for .NET

DisassociateAddress

次の例は、DisassociateAddress を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
/// Disassociate an Elastic IP address from an EC2 instance.
/// </summary>
/// <param name="associationId">The association Id.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DisassociateIp(string associationId)
{
    var response = await _amazonEC2.DisassociateAddressAsync(
        new DisassociateAddressRequest { AssociationId = associationId });
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- API の詳細については、「 API リファレンスDisassociateAddress」の「」を参照してください。 AWS SDK for .NET

`RebootInstances`

次の例は、RebootInstances を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
/// Reboot EC2 instances.
/// </summary>
```

```
   /// <param name="ec2InstanceId">The instance Id of the instances that will be
rebooted.</param>
   /// <returns>Async task.</returns>
   public async Task RebootInstances(string ec2InstanceId)
   {
       var request = new RebootInstancesRequest
       {
           InstanceIds = new List<string> { ec2InstanceId },
       };

       var response = await _amazonEC2.RebootInstancesAsync(request);
       if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
       {
           Console.WriteLine("Instances successfully rebooted.");
       }
       else
       {
           Console.WriteLine("Could not reboot one or more instances.");
       }
   }
```

インスタンスのプロファイルを置き換えて再起動し、ウェブサーバーを再起動します。

```
   /// <summary>
   /// Replace the profile associated with a running instance. After the profile is
replaced, the instance
   /// is rebooted to ensure that it uses the new profile. When the instance is
ready, Systems Manager is
   /// used to restart the Python web server.
   /// </summary>
   /// <param name="instanceId">The Id of the instance to update.</param>
   /// <param name="credsProfileName">The name of the new profile to associate with
the specified instance.</param>
   /// <param name="associationId">The Id of the existing profile association for
the instance.</param>
   /// <returns>Async task.</returns>
   public async Task ReplaceInstanceProfile(string instanceId, string
credsProfileName, string associationId)
   {
       await _amazonEc2.ReplaceIamInstanceProfileAssociationAsync(
               new ReplaceIamInstanceProfileAssociationRequest()
               {
```

```
                    AssociationId = associationId,
                    IamInstanceProfile = new IamInstanceProfileSpecification()
                    {
                        Name = credsProfileName
                    }
                });
        // Allow time before resetting.
        Thread.Sleep(25000);
        var instanceReady = false;
        var retries = 5;
        while (retries-- > 0 && !instanceReady)
        {
            await _amazonEc2.RebootInstancesAsync(
                new RebootInstancesRequest(new List<string>() { instanceId }));
            Thread.Sleep(10000);

            var instancesPaginator =
_amazonSsm.Paginators.DescribeInstanceInformation(
                new DescribeInstanceInformationRequest());
            // Get the entire list using the paginator.
            await foreach (var instance in
instancesPaginator.InstanceInformationList)
            {
                instanceReady = instance.InstanceId == instanceId;
                if (instanceReady)
                {
                    break;
                }
            }
        }
        Console.WriteLine($"Sending restart command to instance {instanceId}");
        await _amazonSsm.SendCommandAsync(
            new SendCommandRequest()
            {
                InstanceIds = new List<string>() { instanceId },
                DocumentName = "AWS-RunShellScript",
                Parameters = new Dictionary<string, List<string>>()
                {
                        {"commands", new List<string>() { "cd / && sudo python3
server.py 80" }}
                }
            });
        Console.WriteLine($"Restarted the web server on instance {instanceId}");
    }
```

- API の詳細については、「 API リファレンスRebootInstances」の「」を参照してください。AWS SDK for .NET

`ReleaseAddress`

次の例は、`ReleaseAddress` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Release an Elastic IP address.
    /// </summary>
    /// <param name="allocationId">The allocation Id of the Elastic IP address.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> ReleaseAddress(string allocationId)
    {
        var request = new ReleaseAddressRequest
        {
            AllocationId = allocationId
        };

        var response = await _amazonEC2.ReleaseAddressAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスReleaseAddress」の「」を参照してください。AWS SDK for .NET

ReplaceIamInstanceProfileAssociation

次の例は、ReplaceIamInstanceProfileAssociation を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
   /// <summary>
   /// Replace the profile associated with a running instance. After the profile is
replaced, the instance
   /// is rebooted to ensure that it uses the new profile. When the instance is
ready, Systems Manager is
   /// used to restart the Python web server.
   /// </summary>
   /// <param name="instanceId">The Id of the instance to update.</param>
   /// <param name="credsProfileName">The name of the new profile to associate with
the specified instance.</param>
   /// <param name="associationId">The Id of the existing profile association for
the instance.</param>
   /// <returns>Async task.</returns>
   public async Task ReplaceInstanceProfile(string instanceId, string
credsProfileName, string associationId)
   {
       await _amazonEc2.ReplaceIamInstanceProfileAssociationAsync(
               new ReplaceIamInstanceProfileAssociationRequest()
               {
                   AssociationId = associationId,
                   IamInstanceProfile = new IamInstanceProfileSpecification()
                   {
                       Name = credsProfileName
                   }
               });
       // Allow time before resetting.
       Thread.Sleep(25000);
       var instanceReady = false;
       var retries = 5;
       while (retries-- > 0 && !instanceReady)
       {
```

```
            await _amazonEc2.RebootInstancesAsync(
                new RebootInstancesRequest(new List<string>() { instanceId }));
            Thread.Sleep(10000);

            var instancesPaginator =
_amazonSsm.Paginators.DescribeInstanceInformation(
                new DescribeInstanceInformationRequest());
            // Get the entire list using the paginator.
            await foreach (var instance in
instancesPaginator.InstanceInformationList)
            {
                instanceReady = instance.InstanceId == instanceId;
                if (instanceReady)
                {
                    break;
                }
            }
        }
        Console.WriteLine($"Sending restart command to instance {instanceId}");
        await _amazonSsm.SendCommandAsync(
            new SendCommandRequest()
            {
                InstanceIds = new List<string>() { instanceId },
                DocumentName = "AWS-RunShellScript",
                Parameters = new Dictionary<string, List<string>>()
                {
                        {"commands", new List<string>() { "cd / && sudo python3
server.py 80" }}
                }
            });
        Console.WriteLine($"Restarted the web server on instance {instanceId}");
    }
```

- API の詳細については、「 API リファレンスReplaceIamInstanceProfileAssociation」の「」を参照してください。 AWS SDK for .NET

RunInstances

次の例は、RunInstances を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Create and run an EC2 instance.
    /// </summary>
    /// <param name="ImageId">The image Id of the image used as a basis for the
    /// EC2 instance.</param>
    /// <param name="instanceType">The instance type of the EC2 instance to
create.</param>
    /// <param name="keyName">The name of the key pair to associate with the
    /// instance.</param>
    /// <param name="groupId">The Id of the Amazon EC2 security group that will be
    /// allowed to interact with the new EC2 instance.</param>
    /// <returns>The instance Id of the new EC2 instance.</returns>
    public async Task<string> RunInstances(string imageId, string instanceType,
string keyName, string groupId)
    {
        var request = new RunInstancesRequest
        {
            ImageId = imageId,
            InstanceType = instanceType,
            KeyName = keyName,
            MinCount = 1,
            MaxCount = 1,
            SecurityGroupIds = new List<string> { groupId }
        };
        var response = await _amazonEC2.RunInstancesAsync(request);
        return response.Reservation.Instances[0].InstanceId;
    }
```

- API の詳細については、「 API リファレンスRunInstances」の「」を参照してください。AWS SDK for .NET

StartInstances

次の例は、StartInstances を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Start an EC2 instance.
    /// </summary>
    /// <param name="ec2InstanceId">The instance Id of the Amazon EC2 instance
    /// to start.</param>
    /// <returns>Async task.</returns>
    public async Task StartInstances(string ec2InstanceId)
    {
        var request = new StartInstancesRequest
        {
            InstanceIds = new List<string> { ec2InstanceId },
        };

        var response = await _amazonEC2.StartInstancesAsync(request);

        if (response.StartingInstances.Count > 0)
        {
            var instances = response.StartingInstances;
            instances.ForEach(i =>
            {
                Console.WriteLine($"Successfully started the EC2 instance with
instance ID: {i.InstanceId}.");
            });
        }
    }
```

- API の詳細については、「 API リファレンスStartInstances」の「」を参照してください。AWS SDK for .NET

StopInstances

次の例は、StopInstances を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
/// Stop an EC2 instance.
/// </summary>
/// <param name="ec2InstanceId">The instance Id of the EC2 instance to
/// stop.</param>
/// <returns>Async task.</returns>
public async Task StopInstances(string ec2InstanceId)
{
    // In addition to the list of instance Ids, the
    // request can also include the following properties:
    //     Force      When true, forces the instances to
    //                stop but you must check the integrity
    //                of the file system. Not recommended on
    //                Windows instances.
    //     Hibernate  When true, hibernates the instance if the
    //                instance was enabled for hibernation when
    //                it was launched.
    var request = new StopInstancesRequest
    {
        InstanceIds = new List<string> { ec2InstanceId },
    };

    var response = await _amazonEC2.StopInstancesAsync(request);

    if (response.StoppingInstances.Count > 0)
    {
        var instances = response.StoppingInstances;
        instances.ForEach(i =>
        {
            Console.WriteLine($"Successfully stopped the EC2 Instance " +
                              $"with InstanceID: {i.InstanceId}.");
```

```
            });
        }
    }
```

- API の詳細については、「 API リファレンスStopInstances」の「」を参照してください。 AWS SDK for .NET

`TerminateInstances`

次の例は、TerminateInstances を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Terminate an EC2 instance.
    /// </summary>
    /// <param name="ec2InstanceId">The instance Id of the EC2 instance
    /// to terminate.</param>
    /// <returns>Async task.</returns>
    public async Task<List<InstanceStateChange>> TerminateInstances(string
ec2InstanceId)
    {
        var request = new TerminateInstancesRequest
        {
            InstanceIds = new List<string> { ec2InstanceId }
        };

        var response = await _amazonEC2.TerminateInstancesAsync(request);
        return response.TerminatingInstances;
    }
```

- API の詳細については、「 API リファレンスTerminateInstances」の「」を参照してください。 AWS SDK for .NET

シナリオ

レジリエントなサービスの構築と管理

次のコード例は、本、映画、曲のレコメンデーションを返す負荷分散型ウェブサービスの作成方法を示しています。この例は、障害に対するサービスの対応方法と、障害発生時の耐障害性を高めるためにサービスを再構築する方法を示しています。

- Amazon EC2 Auto Scaling グループを使用して、起動テンプレートに基づいて Amazon Elastic Compute Cloud (Amazon EC2) インスタンスを作成し、インスタンス数を所定の範囲内に維持します。
- Elastic Load Balancing で HTTP リクエストを処理して配信します。
- Auto Scaling グループ内のインスタンスの状態を監視し、正常なインスタンスにのみリクエストを転送します。
- 各 EC2 インスタンスで Python ウェブサーバーを実行して HTTP リクエストを処理します。ウェブサーバーはレコメンデーションとヘルスチェックを返します。
- Amazon DynamoDB テーブルを使用してレコメンデーションサービスをシミュレートできます。
- AWS Systems Manager パラメータを更新して、リクエストとヘルスチェックに対するウェブサーバーの応答を制御します。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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のシナリオを実行します。

```
static async Task Main(string[] args)
{
    _configuration = new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("settings.json") // Load settings from .json file.
        .AddJsonFile("settings.local.json",
            true) // Optionally, load local settings.
        .Build();
```

```
        // Set up dependency injection for the AWS services.
        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<IAmazonIdentityManagementService>()
                    .AddAWSService<IAmazonDynamoDB>()
                    .AddAWSService<IAmazonElasticLoadBalancingV2>()
                    .AddAWSService<IAmazonSimpleSystemsManagement>()
                    .AddAWSService<IAmazonAutoScaling>()
                    .AddAWSService<IAmazonEC2>()
                    .AddTransient<AutoScalerWrapper>()
                    .AddTransient<ElasticLoadBalancerWrapper>()
                    .AddTransient<SmParameterWrapper>()
                    .AddTransient<Recommendations>()
                    .AddSingleton<IConfiguration>(_configuration)
            )
            .Build();

        ServicesSetup(host);
        ResourcesSetup();

        try
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Welcome to the Resilient Architecture Example
Scenario.");
            Console.WriteLine(new string('-', 80));
            await Deploy(true);

            Console.WriteLine("Now let's begin the scenario.");
            Console.WriteLine(new string('-', 80));
            await Demo(true);

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Finally, let's clean up our resources.");
            Console.WriteLine(new string('-', 80));

            await DestroyResources(true);
```

```
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Resilient Architecture Example Scenario is
complete.");
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
            await DestroyResources(true);
            Console.WriteLine(new string('-', 80));
        }
    }

    /// <summary>
    /// Setup any common resources, also used for integration testing.
    /// </summary>
    public static void ResourcesSetup()
    {
        _httpClient = new HttpClient();
    }

    /// <summary>
    /// Populate the services for use within the console application.
    /// </summary>
    /// <param name="host">The services host.</param>
    private static void ServicesSetup(IHost host)
    {
        _elasticLoadBalancerWrapper =
host.Services.GetRequiredService<ElasticLoadBalancerWrapper>();
        _iamClient =
host.Services.GetRequiredService<IAmazonIdentityManagementService>();
        _recommendations = host.Services.GetRequiredService<Recommendations>();
        _autoScalerWrapper = host.Services.GetRequiredService<AutoScalerWrapper>();
        _smParameterWrapper =
host.Services.GetRequiredService<SmParameterWrapper>();
    }

    /// <summary>
    /// Deploy necessary resources for the scenario.
    /// </summary>
    /// <param name="interactive">True to run as interactive.</param>
```

```
   /// <returns>True if successful.</returns>
   public static async Task<bool> Deploy(bool interactive)
   {
       var protocol = "HTTP";
       var port = 80;
       var sshPort = 22;

       Console.WriteLine(
           "\nFor this demo, we'll use the AWS SDK for .NET to create several AWS
resources\n" +
           "to set up a load-balanced web service endpoint and explore some ways to
make it resilient\n" +
           "against various kinds of failures.\n\n" +
           "Some of the resources create by this demo are:\n");

       Console.WriteLine(
           "\t* A DynamoDB table that the web service depends on to provide book,
movie, and song recommendations.");
       Console.WriteLine(
           "\t* An EC2 launch template that defines EC2 instances that each contain
a Python web server.");
       Console.WriteLine(
           "\t* An EC2 Auto Scaling group that manages EC2 instances across several
Availability Zones.");
       Console.WriteLine(
           "\t* An Elastic Load Balancing (ELB) load balancer that targets the Auto
Scaling group to distribute requests.");
       Console.WriteLine(new string('-', 80));
       Console.WriteLine("Press Enter when you're ready to start deploying
resources.");
       if (interactive)
           Console.ReadLine();

       // Create and populate the DynamoDB table.
       var databaseTableName = _configuration["databaseName"];
       var recommendationsPath = Path.Join(_configuration["resourcePath"],
           "recommendations_objects.json");
       Console.WriteLine($"Creating and populating a DynamoDB table named
{databaseTableName}.");
       await _recommendations.CreateDatabaseWithName(databaseTableName);
       await _recommendations.PopulateDatabase(databaseTableName,
recommendationsPath);
       Console.WriteLine(new string('-', 80));
```

```
        // Create the EC2 Launch Template.

        Console.WriteLine(
            $"Creating an EC2 launch template that runs 'server_startup_script.sh'
when an instance starts.\n"
            + "\nThis script starts a Python web server defined in the `server.py`
script. The web server\n"
            + "listens to HTTP requests on port 80 and responds to requests to '/'
and to '/healthcheck'.\n"
            + "For demo purposes, this server is run as the root user. In
production, the best practice is to\n"
            + "run a web server, such as Apache, with least-privileged
credentials.");
        Console.WriteLine(
            "\nThe template also defines an IAM policy that each instance uses to
assume a role that grants\n"
            + "permissions to access the DynamoDB recommendation table and Systems
Manager parameters\n"
            + "that control the flow of the demo.");

        var startupScriptPath = Path.Join(_configuration["resourcePath"],
            "server_startup_script.sh");
        var instancePolicyPath = Path.Join(_configuration["resourcePath"],
            "instance_policy.json");
        await _autoScalerWrapper.CreateTemplate(startupScriptPath,
instancePolicyPath);
        Console.WriteLine(new string('-', 80));

        Console.WriteLine(
            "Creating an EC2 Auto Scaling group that maintains three EC2 instances,
each in a different\n"
            + "Availability Zone.\n");
        var zones = await _autoScalerWrapper.DescribeAvailabilityZones();
        await _autoScalerWrapper.CreateGroupOfSize(3, _autoScalerWrapper.GroupName,
zones);
        Console.WriteLine(new string('-', 80));

        Console.WriteLine(
            "At this point, you have EC2 instances created. Once each instance
starts, it listens for\n"
            + "HTTP requests. You can see these instances in the console or continue
with the demo.\n");

        Console.WriteLine(new string('-', 80));
```

```
        Console.WriteLine("Press Enter when you're ready to continue.");
        if (interactive)
            Console.ReadLine();

        Console.WriteLine("Creating variables that control the flow of the demo.");
        await _smParameterWrapper.Reset();

        Console.WriteLine(
            "\nCreating an Elastic Load Balancing target group and load balancer.
 The target group\n"
            + "defines how the load balancer connects to instances. The load
 balancer provides a\n"
            + "single endpoint where clients connect and dispatches requests to
 instances in the group.");

        var defaultVpc = await _autoScalerWrapper.GetDefaultVpc();
        var subnets = await
 _autoScalerWrapper.GetAllVpcSubnetsForZones(defaultVpc.VpcId, zones);
        var subnetIds = subnets.Select(s => s.SubnetId).ToList();
        var targetGroup = await
 _elasticLoadBalancerWrapper.CreateTargetGroupOnVpc(_elasticLoadBalancerWrapper.TargetGroupN
 protocol, port, defaultVpc.VpcId);

        await
 _elasticLoadBalancerWrapper.CreateLoadBalancerAndListener(_elasticLoadBalancerWrapper.LoadB
 subnetIds, targetGroup);
        await
 _autoScalerWrapper.AttachLoadBalancerToGroup(_autoScalerWrapper.GroupName,
 targetGroup.TargetGroupArn);
        Console.WriteLine("\nVerifying access to the load balancer endpoint...");
        var endPoint = await
 _elasticLoadBalancerWrapper.GetEndpointForLoadBalancerByName(_elasticLoadBalancerWrapper.Lo
        var loadBalancerAccess = await
 _elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);

        if (!loadBalancerAccess)
        {
            Console.WriteLine("\nCouldn't connect to the load balancer, verifying
 that the port is open...");

            var ipString = await _httpClient.GetStringAsync("https://
checkip.amazonaws.com");
            ipString = ipString.Trim();
```

```
            var defaultSecurityGroup = await
 _autoScalerWrapper.GetDefaultSecurityGroupForVpc(defaultVpc);
            var portIsOpen =
 _autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, port, ipString);
            var sshPortIsOpen =
 _autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, sshPort,
 ipString);

            if (!portIsOpen)
            {
                Console.WriteLine(
                    "\nFor this example to work, the default security group for your
 default VPC must\n"
                    + "allows access from this computer. You can either add it
 automatically from this\n"
                    + "example or add it yourself using the AWS Management Console.
\n");

                if (!interactive || GetYesNoResponse(
                        "Do you want to add a rule to the security group to allow
 inbound traffic from your computer's IP address?"))
                {
                    await
 _autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, port, ipString);
                }
            }

            if (!sshPortIsOpen)
            {
                if (!interactive || GetYesNoResponse(
                        "Do you want to add a rule to the security group to allow
 inbound SSH traffic for debugging from your computer's IP address?"))
                {
                    await
 _autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, sshPort,
 ipString);
                }
            }
            loadBalancerAccess = await
 _elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);
        }

        if (loadBalancerAccess)
        {
```

```
            Console.WriteLine("Your load balancer is ready. You can access it by
browsing to:");
            Console.WriteLine($"\thttp://{endPoint}\n");
        }
        else
        {
            Console.WriteLine(
                "\nCouldn't get a successful response from the load balancer
endpoint. Troubleshoot by\n"
                + "manually verifying that your VPC and security group are
configured correctly and that\n"
                + "you can successfully make a GET request to the load balancer
endpoint:\n");
            Console.WriteLine($"\thttp://{endPoint}\n");
        }
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to continue with the
demo.");
        if (interactive)
            Console.ReadLine();
        return true;
    }

    /// <summary>
    /// Demonstrate the steps of the scenario.
    /// </summary>
    /// <param name="interactive">True to run as an interactive scenario.</param>
    /// <returns>Async task.</returns>
    public static async Task<bool> Demo(bool interactive)
    {
        var ssmOnlyPolicy = Path.Join(_configuration["resourcePath"],
            "ssm_only_policy.json");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Resetting parameters to starting values for demo.");
        await _smParameterWrapper.Reset();

        Console.WriteLine("\nThis part of the demonstration shows how to toggle
different parts of the system\n" +
                          "to create situations where the web service fails, and
shows how using a resilient\n" +
                          "architecture can keep the web service running in spite of
these failures.");
        Console.WriteLine(new string('-', 88));
```

```
        Console.WriteLine("At the start, the load balancer endpoint returns
 recommendations and reports that all targets are healthy.");
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine($"The web service running on the EC2 instances gets
 recommendations by querying a DynamoDB table.\n" +
                          $"The table name is contained in a Systems Manager
 parameter named '{_smParameterWrapper.TableParameter}'.\n" +
                          $"To simulate a failure of the recommendation service,
 let's set this parameter to name a non-existent table.\n");
        await
 _smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter, "this-
is-not-a-table");
        Console.WriteLine("\nNow, sending a GET request to the load balancer
 endpoint returns a failure code. But, the service reports as\n" +
                          "healthy to the load balancer because shallow health
 checks don't check for failure of the recommendation service.");
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("Instead of failing when the recommendation service fails,
 the web service can return a static response.");
        Console.WriteLine("While this is not a perfect solution, it presents the
 customer with a somewhat better experience than failure.");

        await
 _smParameterWrapper.PutParameterByName(_smParameterWrapper.FailureResponseParameter,
 "static");

        Console.WriteLine("\nNow, sending a GET request to the load balancer
 endpoint returns a static response.");
        Console.WriteLine("The service still reports as healthy because health
 checks are still shallow.");
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("Let's reinstate the recommendation service.\n");
        await
 _smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter,
 _smParameterWrapper.TableName);
        Console.WriteLine(
            "\nLet's also substitute bad credentials for one of the instances in the
 target group so that it can't\n" +
```

```
            "access the DynamoDB recommendation table.\n"
        );
        await _autoScalerWrapper.CreateInstanceProfileWithName(
            _autoScalerWrapper.BadCredsPolicyName,
            _autoScalerWrapper.BadCredsRoleName,
            _autoScalerWrapper.BadCredsProfileName,
            ssmOnlyPolicy,
            new List<string> { "AmazonSSMManagedInstanceCore" }
        );
        var instances = await
_autoScalerWrapper.GetInstancesByGroupName(_autoScalerWrapper.GroupName);
        var badInstanceId = instances.First();
        var instanceProfile = await
_autoScalerWrapper.GetInstanceProfile(badInstanceId);
        Console.WriteLine(
            $"Replacing the profile for instance {badInstanceId} with a profile that
contains\n" +
            "bad credentials...\n"
        );
        await _autoScalerWrapper.ReplaceInstanceProfile(
            badInstanceId,
            _autoScalerWrapper.BadCredsProfileName,
            instanceProfile.AssociationId
        );
        Console.WriteLine(
            "Now, sending a GET request to the load balancer endpoint returns either
a recommendation or a static response,\n" +
            "depending on which instance is selected by the load balancer.\n"
        );
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nLet's implement a deep health check. For this demo, a
deep health check tests whether");
        Console.WriteLine("the web service can access the DynamoDB table that it
depends on for recommendations. Note that");
        Console.WriteLine("the deep health check is only for ELB routing and not for
Auto Scaling instance health.");
        Console.WriteLine("This kind of deep health check is not recommended for
Auto Scaling instance health, because it");
        Console.WriteLine("risks accidental termination of all instances in the Auto
Scaling group when a dependent service fails.");
```

```
        Console.WriteLine("\nBy implementing deep health checks, the load balancer
can detect when one of the instances is failing");
        Console.WriteLine("and take that instance out of rotation.");

        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.HealthCheckParameter,
"deep");

        Console.WriteLine($"\nNow, checking target health indicates that the
instance with bad credentials ({badInstanceId})");
        Console.WriteLine("is unhealthy. Note that it might take a minute or two for
the load balancer to detect the unhealthy");
        Console.WriteLine("instance. Sending a GET request to the load balancer
endpoint always returns a recommendation, because");
        Console.WriteLine("the load balancer takes unhealthy instances out of its
rotation.");

        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nBecause the instances in this demo are controlled by an
auto scaler, the simplest way to fix an unhealthy");
        Console.WriteLine("instance is to terminate it and let the auto scaler start
a new instance to replace it.");

        await _autoScalerWrapper.TryTerminateInstanceById(badInstanceId);

        Console.WriteLine($"\nEven while the instance is terminating and the new
instance is starting, sending a GET");
        Console.WriteLine("request to the web service continues to get a successful
recommendation response because");
        Console.WriteLine("starts and reports as healthy, it is included in the load
balancing rotation.");
        Console.WriteLine("Note that terminating and replacing an instance typically
takes several minutes, during which time you");
        Console.WriteLine("can see the changing health check status until the new
instance is running and healthy.");

        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nIf the recommendation service fails now, deep health
checks mean all instances report as unhealthy.");
```

```
        await
 _smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter, "this-
is-not-a-table");

        Console.WriteLine($"\nWhen all instances are unhealthy, the load balancer
 continues to route requests even to");
        Console.WriteLine("unhealthy instances, allowing them to fail open and
 return a static response rather than fail");
        Console.WriteLine("closed and report failure to the customer.");

        if (interactive)
            await DemoActionChoices();
        await _smParameterWrapper.Reset();

        Console.WriteLine(new string('-', 80));
        return true;
    }

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <param name="interactive">True to ask the user for cleanup.</param>
    /// <returns>Async task.</returns>
    public static async Task<bool> DestroyResources(bool interactive)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine(
            "To keep things tidy and to avoid unwanted charges on your account, we
 can clean up all AWS resources\n" +
            "that were created for this demo."
        );

        if (!interactive || GetYesNoResponse("Do you want to clean up all demo
 resources? (y/n) "))
        {
            await
 _elasticLoadBalancerWrapper.DeleteLoadBalancerByName(_elasticLoadBalancerWrapper.LoadBalanc
            await
 _elasticLoadBalancerWrapper.DeleteTargetGroupByName(_elasticLoadBalancerWrapper.TargetGroup
            await
 _autoScalerWrapper.TerminateAndDeleteAutoScalingGroupWithName(_autoScalerWrapper.GroupName)
            await
 _autoScalerWrapper.DeleteKeyPairByName(_autoScalerWrapper.KeyPairName);
```

```
            await
_autoScalerWrapper.DeleteTemplateByName(_autoScalerWrapper.LaunchTemplateName);
            await _autoScalerWrapper.DeleteInstanceProfile(
                _autoScalerWrapper.BadCredsProfileName,
                _autoScalerWrapper.BadCredsRoleName
            );
            await
_recommendations.DestroyDatabaseByName(_recommendations.TableName);
        }
        else
        {
            Console.WriteLine(
                "Ok, we'll leave the resources intact.\n" +
                "Don't forget to delete them when you're done with them or you might
incur unexpected charges."
            );
        }

        Console.WriteLine(new string('-', 80));
        return true;
    }
```

Auto Scaling と Amazon EC2 のアクションをラップするクラスを作成します。

```
/// <summary>
/// Encapsulates Amazon EC2 Auto Scaling and EC2 management methods.
/// </summary>
public class AutoScalerWrapper
{
    private readonly IAmazonAutoScaling _amazonAutoScaling;
    private readonly IAmazonEC2 _amazonEc2;
    private readonly IAmazonSimpleSystemsManagement _amazonSsm;
    private readonly IAmazonIdentityManagementService _amazonIam;

    private readonly string _instanceType = "";
    private readonly string _amiParam = "";
    private readonly string _launchTemplateName = "";
    private readonly string _groupName = "";
    private readonly string _instancePolicyName = "";
    private readonly string _instanceRoleName = "";
    private readonly string _instanceProfileName = "";
    private readonly string _badCredsProfileName = "";
```

```
    private readonly string _badCredsRoleName = "";
    private readonly string _badCredsPolicyName = "";
    private readonly string _keyPairName = "";

    public string GroupName => _groupName;
    public string KeyPairName => _keyPairName;
    public string LaunchTemplateName => _launchTemplateName;
    public string InstancePolicyName => _instancePolicyName;
    public string BadCredsProfileName => _badCredsProfileName;
    public string BadCredsRoleName => _badCredsRoleName;
    public string BadCredsPolicyName => _badCredsPolicyName;

    /// <summary>
    /// Constructor for the AutoScalerWrapper.
    /// </summary>
    /// <param name="amazonAutoScaling">The injected AutoScaling client.</param>
    /// <param name="amazonEc2">The injected EC2 client.</param>
    /// <param name="amazonIam">The injected IAM client.</param>
    /// <param name="amazonSsm">The injected SSM client.</param>
    public AutoScalerWrapper(
        IAmazonAutoScaling amazonAutoScaling,
        IAmazonEC2 amazonEc2,
        IAmazonSimpleSystemsManagement amazonSsm,
        IAmazonIdentityManagementService amazonIam,
        IConfiguration configuration)
    {
        _amazonAutoScaling = amazonAutoScaling;
        _amazonEc2 = amazonEc2;
        _amazonSsm = amazonSsm;
        _amazonIam = amazonIam;

        var prefix = configuration["resourcePrefix"];
        _instanceType = configuration["instanceType"];
        _amiParam = configuration["amiParam"];

        _launchTemplateName = prefix + "-template";
        _groupName = prefix + "-group";
        _instancePolicyName = prefix + "-pol";
        _instanceRoleName = prefix + "-role";
        _instanceProfileName = prefix + "-prof";
        _badCredsPolicyName = prefix + "-bc-pol";
        _badCredsRoleName = prefix + "-bc-role";
        _badCredsProfileName = prefix + "-bc-prof";
        _keyPairName = prefix + "-key-pair";
```

```
    }

    /// <summary>
    /// Create a policy, role, and profile that is associated with instances with a
specified name.
    /// An instance's associated profile defines a role that is assumed by the
    /// instance.The role has attached policies that specify the AWS permissions
granted to
    /// clients that run on the instance.
    /// </summary>
    /// <param name="policyName">Name to use for the policy.</param>
    /// <param name="roleName">Name to use for the role.</param>
    /// <param name="profileName">Name to use for the profile.</param>
    /// <param name="ssmOnlyPolicyFile">Path to a policy file for SSM.</param>
    /// <param name="awsManagedPolicies">AWS Managed policies to be attached to the
role.</param>
    /// <returns>The Arn of the profile.</returns>
    public async Task<string> CreateInstanceProfileWithName(
        string policyName,
        string roleName,
        string profileName,
        string ssmOnlyPolicyFile,
        List<string>? awsManagedPolicies = null)
    {

        var assumeRoleDoc = "{" +
                                    "\"Version\": \"2012-10-17\"," +
                                    "\"Statement\": [{" +
                                         "\"Effect\": \"Allow\"," +
                                         "\"Principal\": {" +
                                         "\"Service\": [" +
                                             "\"ec2.amazonaws.com\"" +
                                         "]" +
                                         "}," +
                                    "\"Action\": \"sts:AssumeRole\"" +
                                    "}]" +
                                "}";

        var policyDocument = await File.ReadAllTextAsync(ssmOnlyPolicyFile);

        var policyArn = "";

        try
        {
```

```
            var createPolicyResult = await _amazonIam.CreatePolicyAsync(
                new CreatePolicyRequest
                {
                    PolicyName = policyName,
                    PolicyDocument = policyDocument
                });
            policyArn = createPolicyResult.Policy.Arn;
        }
        catch (EntityAlreadyExistsException)
        {
            // The policy already exists, so we look it up to get the Arn.
            var policiesPaginator = _amazonIam.Paginators.ListPolicies(
                new ListPoliciesRequest()
                {
                    Scope = PolicyScopeType.Local
                });
            // Get the entire list using the paginator.
            await foreach (var policy in policiesPaginator.Policies)
            {
                if (policy.PolicyName.Equals(policyName))
                {
                    policyArn = policy.Arn;
                }
            }

            if (policyArn == null)
            {
                throw new InvalidOperationException("Policy not found");
            }
        }

        try
        {
            await _amazonIam.CreateRoleAsync(new CreateRoleRequest()
            {
                RoleName = roleName,
                AssumeRolePolicyDocument = assumeRoleDoc,
            });
            await _amazonIam.AttachRolePolicyAsync(new AttachRolePolicyRequest()
            {
                RoleName = roleName,
                PolicyArn = policyArn
            });
            if (awsManagedPolicies != null)
```

```
            {
                foreach (var awsPolicy in awsManagedPolicies)
                {
                    await _amazonIam.AttachRolePolicyAsync(new
AttachRolePolicyRequest()
                    {
                        PolicyArn = $"arn:aws:iam::aws:policy/{awsPolicy}",
                        RoleName = roleName
                    });
                }
            }
        }
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine("Role already exists.");
        }

        string profileArn = "";
        try
        {
            var profileCreateResponse = await _amazonIam.CreateInstanceProfileAsync(
                new CreateInstanceProfileRequest()
                {
                    InstanceProfileName = profileName
                });
            // Allow time for the profile to be ready.
            profileArn = profileCreateResponse.InstanceProfile.Arn;
            Thread.Sleep(10000);
            await _amazonIam.AddRoleToInstanceProfileAsync(
                new AddRoleToInstanceProfileRequest()
                {
                    InstanceProfileName = profileName,
                    RoleName = roleName
                });

        }
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine("Policy already exists.");
            var profileGetResponse = await _amazonIam.GetInstanceProfileAsync(
                new GetInstanceProfileRequest()
                {
                    InstanceProfileName = profileName
                });
```

```
            profileArn = profileGetResponse.InstanceProfile.Arn;
        }
        return profileArn;
    }

    /// <summary>
    /// Create a new key pair and save the file.
    /// </summary>
    /// <param name="newKeyPairName">The name of the new key pair.</param>
    /// <returns>Async task.</returns>
    public async Task CreateKeyPair(string newKeyPairName)
    {
        try
        {
            var keyResponse = await _amazonEc2.CreateKeyPairAsync(
                new CreateKeyPairRequest() { KeyName = newKeyPairName });
            await File.WriteAllTextAsync($"{newKeyPairName}.pem",
                keyResponse.KeyPair.KeyMaterial);
            Console.WriteLine($"Created key pair {newKeyPairName}.");
        }
        catch (AlreadyExistsException)
        {
            Console.WriteLine("Key pair already exists.");
        }
    }

    /// <summary>
    /// Delete the key pair and file by name.
    /// </summary>
    /// <param name="deleteKeyPairName">The key pair to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteKeyPairByName(string deleteKeyPairName)
    {
        try
        {
            await _amazonEc2.DeleteKeyPairAsync(
                new DeleteKeyPairRequest() { KeyName = deleteKeyPairName });
            File.Delete($"{deleteKeyPairName}.pem");
        }
        catch (FileNotFoundException)
        {
            Console.WriteLine($"Key pair {deleteKeyPairName} not found.");
        }
    }
```

```
   /// <summary>
   /// Creates an Amazon EC2 launch template to use with Amazon EC2 Auto Scaling.
   /// The launch template specifies a Bash script in its user data field that runs
after
   /// the instance is started. This script installs the Python packages and starts
a Python
   /// web server on the instance.
   /// </summary>
   /// <param name="startupScriptPath">The path to a Bash script file that is
run.</param>
   /// <param name="instancePolicyPath">The path to a permissions policy to create
and attach to the profile.</param>
   /// <returns>The template object.</returns>
   public async Task<Amazon.EC2.Model.LaunchTemplate> CreateTemplate(string
startupScriptPath, string instancePolicyPath)
   {
       await CreateKeyPair(_keyPairName);
       await CreateInstanceProfileWithName(_instancePolicyName, _instanceRoleName,
_instanceProfileName, instancePolicyPath);

       var startServerText = await File.ReadAllTextAsync(startupScriptPath);
       var plainTextBytes = System.Text.Encoding.UTF8.GetBytes(startServerText);

       var amiLatest = await _amazonSsm.GetParameterAsync(
           new GetParameterRequest() { Name = _amiParam });
       var amiId = amiLatest.Parameter.Value;
       var launchTemplateResponse = await _amazonEc2.CreateLaunchTemplateAsync(
           new CreateLaunchTemplateRequest()
           {
               LaunchTemplateName = _launchTemplateName,
               LaunchTemplateData = new RequestLaunchTemplateData()
               {
                   InstanceType = _instanceType,
                   ImageId = amiId,
                   IamInstanceProfile =
                       new
                           LaunchTemplateIamInstanceProfileSpecificationRequest()
                       {
                           Name = _instanceProfileName
                       },
                   KeyName = _keyPairName,
                   UserData = System.Convert.ToBase64String(plainTextBytes)
               }
```

```
            });
        return launchTemplateResponse.LaunchTemplate;

    }

    /// <summary>
    /// Get a list of Availability Zones in the AWS Region of the Amazon EC2 Client.
    /// </summary>
    /// <returns>A list of availability zones.</returns>
    public async Task<List<string>> DescribeAvailabilityZones()
    {
        var zoneResponse = await _amazonEc2.DescribeAvailabilityZonesAsync(
            new DescribeAvailabilityZonesRequest());
        return zoneResponse.AvailabilityZones.Select(z => z.ZoneName).ToList();
    }

    /// <summary>
    /// Create an EC2 Auto Scaling group of a specified size and name.
    /// </summary>
    /// <param name="groupSize">The size for the group.</param>
    /// <param name="groupName">The name for the group.</param>
    /// <param name="availabilityZones">The availability zones for the group.</
param>
    /// <returns>Async task.</returns>
    public async Task CreateGroupOfSize(int groupSize, string groupName,
 List<string> availabilityZones)
    {
        try
        {
            await _amazonAutoScaling.CreateAutoScalingGroupAsync(
                new CreateAutoScalingGroupRequest()
                {
                    AutoScalingGroupName = groupName,
                    AvailabilityZones = availabilityZones,
                    LaunchTemplate =
                        new Amazon.AutoScaling.Model.LaunchTemplateSpecification()
                        {
                            LaunchTemplateName = _launchTemplateName,
                            Version = "$Default"
                        },
                    MaxSize = groupSize,
                    MinSize = groupSize
                });
```

```
            Console.WriteLine($"Created EC2 Auto Scaling group {groupName} with size
{groupSize}.");
        }
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine($"EC2 Auto Scaling group {groupName} already
exists.");
        }
    }

    /// <summary>
    /// Get the default VPC for the account.
    /// </summary>
    /// <returns>The default VPC object.</returns>
    public async Task<Vpc> GetDefaultVpc()
    {
        var vpcResponse = await _amazonEc2.DescribeVpcsAsync(
            new DescribeVpcsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("is-default", new List<string>() { "true" })
                }
            });
        return vpcResponse.Vpcs[0];
    }

    /// <summary>
    /// Get all the subnets for a Vpc in a set of availability zones.
    /// </summary>
    /// <param name="vpcId">The Id of the Vpc.</param>
    /// <param name="availabilityZones">The list of availability zones.</param>
    /// <returns>The collection of subnet objects.</returns>
    public async Task<List<Subnet>> GetAllVpcSubnetsForZones(string vpcId,
List<string> availabilityZones)
    {
        var subnets = new List<Subnet>();
        var subnetPaginator = _amazonEc2.Paginators.DescribeSubnets(
            new DescribeSubnetsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("vpc-id", new List<string>() { vpcId}),
                    new ("availability-zone", availabilityZones),
```

```
                    new ("default-for-az", new List<string>() { "true" })
                }
            });

        // Get the entire list using the paginator.
        await foreach (var subnet in subnetPaginator.Subnets)
        {
            subnets.Add(subnet);
        }

        return subnets;
    }

    /// <summary>
    /// Delete a launch template by name.
    /// </summary>
    /// <param name="templateName">The name of the template to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteTemplateByName(string templateName)
    {
        try
        {
            await _amazonEc2.DeleteLaunchTemplateAsync(
                new DeleteLaunchTemplateRequest()
                {
                    LaunchTemplateName = templateName
                });
        }
        catch (AmazonClientException)
        {
            Console.WriteLine($"Unable to delete template {templateName}.");
        }
    }

    /// <summary>
    /// Detaches a role from an instance profile, detaches policies from the role,
    /// and deletes all the resources.
    /// </summary>
    /// <param name="profileName">The name of the profile to delete.</param>
    /// <param name="roleName">The name of the role to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteInstanceProfile(string profileName, string roleName)
    {
        try
```

```
        {
            await _amazonIam.RemoveRoleFromInstanceProfileAsync(
                new RemoveRoleFromInstanceProfileRequest()
                {
                    InstanceProfileName = profileName,
                    RoleName = roleName
                });
            await _amazonIam.DeleteInstanceProfileAsync(
                new DeleteInstanceProfileRequest() { InstanceProfileName =
profileName });
            var attachedPolicies = await _amazonIam.ListAttachedRolePoliciesAsync(
                new ListAttachedRolePoliciesRequest() { RoleName = roleName });
            foreach (var policy in attachedPolicies.AttachedPolicies)
            {
                await _amazonIam.DetachRolePolicyAsync(
                    new DetachRolePolicyRequest()
                    {
                        RoleName = roleName,
                        PolicyArn = policy.PolicyArn
                    });
                // Delete the custom policies only.
                if (!policy.PolicyArn.StartsWith("arn:aws:iam::aws"))
                {
                    await _amazonIam.DeletePolicyAsync(
                        new Amazon.IdentityManagement.Model.DeletePolicyRequest()
                        {
                            PolicyArn = policy.PolicyArn
                        });
                }
            }

            await _amazonIam.DeleteRoleAsync(
                new DeleteRoleRequest() { RoleName = roleName });
        }
        catch (NoSuchEntityException)
        {
            Console.WriteLine($"Instance profile {profileName} does not exist.");
        }
    }

    /// <summary>
    /// Gets data about the instances in an EC2 Auto Scaling group by its group
name.
    /// </summary>
```

```
    /// <param name="group">The name of the auto scaling group.</param>
    /// <returns>A collection of instance Ids.</returns>
    public async Task<IEnumerable<string>> GetInstancesByGroupName(string group)
    {
        var instanceResponse = await
_amazonAutoScaling.DescribeAutoScalingGroupsAsync(
            new DescribeAutoScalingGroupsRequest()
            {
                AutoScalingGroupNames = new List<string>() { group }
            });
        var instanceIds = instanceResponse.AutoScalingGroups.SelectMany(
            g => g.Instances.Select(i => i.InstanceId));
        return instanceIds;
    }

    /// <summary>
    /// Get the instance profile association data for an instance.
    /// </summary>
    /// <param name="instanceId">The Id of the instance.</param>
    /// <returns>Instance profile associations data.</returns>
    public async Task<IamInstanceProfileAssociation> GetInstanceProfile(string
instanceId)
    {
        var response = await _amazonEc2.DescribeIamInstanceProfileAssociationsAsync(
            new DescribeIamInstanceProfileAssociationsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("instance-id", new List<string>() { instanceId })
                },
            });
        return response.IamInstanceProfileAssociations[0];
    }

    /// <summary>
    /// Replace the profile associated with a running instance. After the profile is
replaced, the instance
    /// is rebooted to ensure that it uses the new profile. When the instance is
ready, Systems Manager is
    /// used to restart the Python web server.
    /// </summary>
    /// <param name="instanceId">The Id of the instance to update.</param>
    /// <param name="credsProfileName">The name of the new profile to associate with
the specified instance.</param>
```

```
   /// <param name="associationId">The Id of the existing profile association for
the instance.</param>
   /// <returns>Async task.</returns>
   public async Task ReplaceInstanceProfile(string instanceId, string
credsProfileName, string associationId)
   {
       await _amazonEc2.ReplaceIamInstanceProfileAssociationAsync(
               new ReplaceIamInstanceProfileAssociationRequest()
               {
                   AssociationId = associationId,
                   IamInstanceProfile = new IamInstanceProfileSpecification()
                   {
                       Name = credsProfileName
                   }
               });
       // Allow time before resetting.
       Thread.Sleep(25000);
       var instanceReady = false;
       var retries = 5;
       while (retries-- > 0 && !instanceReady)
       {
           await _amazonEc2.RebootInstancesAsync(
               new RebootInstancesRequest(new List<string>() { instanceId }));
           Thread.Sleep(10000);

           var instancesPaginator =
_amazonSsm.Paginators.DescribeInstanceInformation(
               new DescribeInstanceInformationRequest());
           // Get the entire list using the paginator.
           await foreach (var instance in
instancesPaginator.InstanceInformationList)
           {
               instanceReady = instance.InstanceId == instanceId;
               if (instanceReady)
               {
                   break;
               }
           }
       }
       Console.WriteLine($"Sending restart command to instance {instanceId}");
       await _amazonSsm.SendCommandAsync(
           new SendCommandRequest()
           {
               InstanceIds = new List<string>() { instanceId },
```

```
                DocumentName = "AWS-RunShellScript",
                Parameters = new Dictionary<string, List<string>>()
                {
                        {"commands", new List<string>() { "cd / && sudo python3
server.py 80" }}
                }
            });
        Console.WriteLine($"Restarted the web server on instance {instanceId}");
    }

    /// <summary>
    /// Try to terminate an instance by its Id.
    /// </summary>
    /// <param name="instanceId">The Id of the instance to terminate.</param>
    /// <returns>Async task.</returns>
    public async Task TryTerminateInstanceById(string instanceId)
    {
        var stopping = false;
        Console.WriteLine($"Stopping {instanceId}...");
        while (!stopping)
        {
            try
            {
                await _amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(
                    new TerminateInstanceInAutoScalingGroupRequest()
                    {
                        InstanceId = instanceId,
                        ShouldDecrementDesiredCapacity = false
                    });
                stopping = true;
            }
            catch (ScalingActivityInProgressException)
            {
                Console.WriteLine($"Scaling activity in progress for {instanceId}.
Waiting...");
                Thread.Sleep(10000);
            }
        }
    }

    /// <summary>
    /// Tries to delete the EC2 Auto Scaling group. If the group is in use or in
progress,
    /// waits and retries until the group is successfully deleted.
```

```
    /// </summary>
    /// <param name="groupName">The name of the group to try to delete.</param>
    /// <returns>Async task.</returns>
    public async Task TryDeleteGroupByName(string groupName)
    {
        var stopped = false;
        while (!stopped)
        {
            try
            {
                await _amazonAutoScaling.DeleteAutoScalingGroupAsync(
                    new DeleteAutoScalingGroupRequest()
                    {
                        AutoScalingGroupName = groupName
                    });
                stopped = true;
            }
            catch (Exception e)
                when ((e is ScalingActivityInProgressException)
                      || (e is Amazon.AutoScaling.Model.ResourceInUseException))
            {
                Console.WriteLine($"Some instances are still running. Waiting...");
                Thread.Sleep(10000);
            }
        }
    }

    /// <summary>
    /// Terminate instances and delete the Auto Scaling group by name.
    /// </summary>
    /// <param name="groupName">The name of the group to delete.</param>
    /// <returns>Async task.</returns>
    public async Task TerminateAndDeleteAutoScalingGroupWithName(string groupName)
    {
        var describeGroupsResponse = await
_amazonAutoScaling.DescribeAutoScalingGroupsAsync(
            new DescribeAutoScalingGroupsRequest()
            {
                AutoScalingGroupNames = new List<string>() { groupName }
            });
        if (describeGroupsResponse.AutoScalingGroups.Any())
        {
            // Update the size to 0.
            await _amazonAutoScaling.UpdateAutoScalingGroupAsync(
```

```
                new UpdateAutoScalingGroupRequest()
                {
                    AutoScalingGroupName = groupName,
                    MinSize = 0
                });
            var group = describeGroupsResponse.AutoScalingGroups[0];
            foreach (var instance in group.Instances)
            {
                await TryTerminateInstanceById(instance.InstanceId);
            }

            await TryDeleteGroupByName(groupName);
        }
        else
        {
            Console.WriteLine($"No groups found with name {groupName}.");
        }
    }


    /// <summary>
    /// Get the default security group for a specified Vpc.
    /// </summary>
    /// <param name="vpc">The Vpc to search.</param>
    /// <returns>The default security group.</returns>
    public async Task<SecurityGroup> GetDefaultSecurityGroupForVpc(Vpc vpc)
    {
        var groupResponse = await _amazonEc2.DescribeSecurityGroupsAsync(
            new DescribeSecurityGroupsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("group-name", new List<string>() { "default" }),
                    new ("vpc-id", new List<string>() { vpc.VpcId })
                }
            });
        return groupResponse.SecurityGroups[0];
    }

    /// <summary>
    /// Verify the default security group of a Vpc allows ingress from the calling
computer.
    /// This can be done by allowing ingress from this computer's IP address.
```

```
   /// In some situations, such as connecting from a corporate network, you must
instead specify
   /// a prefix list Id. You can also temporarily open the port to any IP address
while running this example.
   /// If you do, be sure to remove public access when you're done.
   /// </summary>
   /// <param name="vpc">The group to check.</param>
   /// <param name="port">The port to verify.</param>
   /// <param name="ipAddress">This computer's IP address.</param>
   /// <returns>True if the ip address is allowed on the group.</returns>
   public bool VerifyInboundPortForGroup(SecurityGroup group, int port, string
ipAddress)
   {
       var portIsOpen = false;
       foreach (var ipPermission in group.IpPermissions)
       {
           if (ipPermission.FromPort == port)
           {
               foreach (var ipRange in ipPermission.Ipv4Ranges)
               {
                   var cidr = ipRange.CidrIp;
                   if (cidr.StartsWith(ipAddress) || cidr == "0.0.0.0/0")
                   {
                       portIsOpen = true;
                   }
               }

               if (ipPermission.PrefixListIds.Any())
               {
                   portIsOpen = true;
               }

               if (!portIsOpen)
               {
                   Console.WriteLine("The inbound rule does not appear to be open
to either this computer's IP\n" +
                                     "address, to all IP addresses (0.0.0.0/0), or
to a prefix list ID.");
               }
               else
               {
                   break;
               }
           }
```

```
        }

        return portIsOpen;
    }

    /// <summary>
    /// Add an ingress rule to the specified security group that allows access on
 the
    /// specified port from the specified IP address.
    /// </summary>
    /// <param name="groupId">The Id of the security group to modify.</param>
    /// <param name="port">The port to open.</param>
    /// <param name="ipAddress">The IP address to allow access.</param>
    /// <returns>Async task.</returns>
    public async Task OpenInboundPort(string groupId, int port, string ipAddress)
    {
        await _amazonEc2.AuthorizeSecurityGroupIngressAsync(
            new AuthorizeSecurityGroupIngressRequest()
            {
                GroupId = groupId,
                IpPermissions = new List<IpPermission>()
                {
                    new IpPermission()
                    {
                        FromPort = port,
                        ToPort = port,
                        IpProtocol = "tcp",
                        Ipv4Ranges = new List<IpRange>()
                        {
                            new IpRange() { CidrIp = $"{ipAddress}/32" }
                        }
                    }
                }
            });
    }

    /// <summary>
    /// Attaches an Elastic Load Balancing (ELB) target group to this EC2 Auto
 Scaling group.
    /// The
    /// </summary>
    /// <param name="autoScalingGroupName">The name of the Auto Scaling group.</
param>
    /// <param name="targetGroupArn">The Arn for the target group.</param>
```

```
    /// <returns>Async task.</returns>
    public async Task AttachLoadBalancerToGroup(string autoScalingGroupName, string
 targetGroupArn)
    {
        await _amazonAutoScaling.AttachLoadBalancerTargetGroupsAsync(
            new AttachLoadBalancerTargetGroupsRequest()
            {
                AutoScalingGroupName = autoScalingGroupName,
                TargetGroupARNs = new List<string>() { targetGroupArn }
            });
    }
}
```

Elastic Load Balancing のアクションをラップするクラスを作成します。

```
/// <summary>
/// Encapsulates Elastic Load Balancer actions.
/// </summary>
public class ElasticLoadBalancerWrapper
{
    private readonly IAmazonElasticLoadBalancingV2 _amazonElasticLoadBalancingV2;
    private string? _endpoint = null;
    private readonly string _targetGroupName = "";
    private readonly string _loadBalancerName = "";
    HttpClient _httpClient = new();

    public string TargetGroupName => _targetGroupName;
    public string LoadBalancerName => _loadBalancerName;

    /// <summary>
    /// Constructor for the Elastic Load Balancer wrapper.
    /// </summary>
    /// <param name="amazonElasticLoadBalancingV2">The injected load balancing v2
 client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public ElasticLoadBalancerWrapper(
        IAmazonElasticLoadBalancingV2 amazonElasticLoadBalancingV2,
        IConfiguration configuration)
    {
        _amazonElasticLoadBalancingV2 = amazonElasticLoadBalancingV2;
        var prefix = configuration["resourcePrefix"];
```

```
        _targetGroupName = prefix + "-tg";
        _loadBalancerName = prefix + "-lb";
    }

    /// <summary>
    /// Get the HTTP Endpoint of a load balancer by its name.
    /// </summary>
    /// <param name="loadBalancerName">The name of the load balancer.</param>
    /// <returns>The HTTP endpoint.</returns>
    public async Task<string> GetEndpointForLoadBalancerByName(string
loadBalancerName)
    {
        if (_endpoint == null)
        {
            var endpointResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                    new DescribeLoadBalancersRequest()
                    {
                        Names = new List<string>() { loadBalancerName }
                    });
            _endpoint = endpointResponse.LoadBalancers[0].DNSName;
        }

        return _endpoint;
    }

    /// <summary>
    /// Return the GET response for an endpoint as text.
    /// </summary>
    /// <param name="endpoint">The endpoint for the request.</param>
    /// <returns>The request response.</returns>
    public async Task<string> GetEndPointResponse(string endpoint)
    {
        var endpointResponse = await _httpClient.GetAsync($"http://{endpoint}");
        var textResponse = await endpointResponse.Content.ReadAsStringAsync();
        return textResponse!;
    }

    /// <summary>
    /// Get the target health for a group by name.
    /// </summary>
    /// <param name="groupName">The name of the group.</param>
    /// <returns>The collection of health descriptions.</returns>
```

```
    public async Task<List<TargetHealthDescription>>
 CheckTargetHealthForGroup(string groupName)
    {
        List<TargetHealthDescription> result = null!;
        try
        {
            var groupResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                    new DescribeTargetGroupsRequest()
                    {
                        Names = new List<string>() { groupName }
                    });
            var healthResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetHealthAsync(
                    new DescribeTargetHealthRequest()
                    {
                        TargetGroupArn =
 groupResponse.TargetGroups[0].TargetGroupArn
                    });
            ;
            result = healthResponse.TargetHealthDescriptions;
        }
        catch (TargetGroupNotFoundException)
        {
            Console.WriteLine($"Target group {groupName} not found.");
        }
        return result;
    }

    /// <summary>
    /// Create an Elastic Load Balancing target group. The target group specifies
 how the load balancer forwards
    /// requests to instances in the group and how instance health is checked.
    ///
    /// To speed up this demo, the health check is configured with shortened times
 and lower thresholds. In production,
    /// you might want to decrease the sensitivity of your health checks to avoid
 unwanted failures.
    /// </summary>
    /// <param name="groupName">The name for the group.</param>
    /// <param name="protocol">The protocol, such as HTTP.</param>
    /// <param name="port">The port to use to forward requests, such as 80.</param>
    /// <param name="vpcId">The Id of the Vpc in which the load balancer exists.</
param>
```

```
   /// <returns>The new TargetGroup object.</returns>
   public async Task<TargetGroup> CreateTargetGroupOnVpc(string groupName,
ProtocolEnum protocol, int port, string vpcId)
   {
       var createResponse = await
_amazonElasticLoadBalancingV2.CreateTargetGroupAsync(
           new CreateTargetGroupRequest()
           {
               Name = groupName,
               Protocol = protocol,
               Port = port,
               HealthCheckPath = "/healthcheck",
               HealthCheckIntervalSeconds = 10,
               HealthCheckTimeoutSeconds = 5,
               HealthyThresholdCount = 2,
               UnhealthyThresholdCount = 2,
               VpcId = vpcId
           });
       var targetGroup = createResponse.TargetGroups[0];
       return targetGroup;
   }

   /// <summary>
   /// Create an Elastic Load Balancing load balancer that uses the specified
subnets
   /// and forwards requests to the specified target group.
   /// </summary>
   /// <param name="name">The name for the new load balancer.</param>
   /// <param name="subnetIds">Subnets for the load balancer.</param>
   /// <param name="targetGroup">Target group for forwarded requests.</param>
   /// <returns>The new LoadBalancer object.</returns>
   public async Task<LoadBalancer> CreateLoadBalancerAndListener(string name,
List<string> subnetIds, TargetGroup targetGroup)
   {
       var createLbResponse = await
_amazonElasticLoadBalancingV2.CreateLoadBalancerAsync(
           new CreateLoadBalancerRequest()
           {
               Name = name,
               Subnets = subnetIds
           });
       var loadBalancerArn = createLbResponse.LoadBalancers[0].LoadBalancerArn;

       // Wait for load balancer to be available.
```

```
        var loadBalancerReady = false;
        while (!loadBalancerReady)
        {
            try
            {
                var describeResponse =
                    await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                        new DescribeLoadBalancersRequest()
                        {
                            Names = new List<string>() { name }
                        });

                var loadBalancerState =
describeResponse.LoadBalancers[0].State.Code;

                loadBalancerReady = loadBalancerState ==
LoadBalancerStateEnum.Active;
            }
            catch (LoadBalancerNotFoundException)
            {
                loadBalancerReady = false;
            }
            Thread.Sleep(10000);
        }
        // Create the listener.
        await _amazonElasticLoadBalancingV2.CreateListenerAsync(
            new CreateListenerRequest()
            {
                LoadBalancerArn = loadBalancerArn,
                Protocol = targetGroup.Protocol,
                Port = targetGroup.Port,
                DefaultActions = new List<Action>()
                {
                    new Action()
                    {
                        Type = ActionTypeEnum.Forward,
                        TargetGroupArn = targetGroup.TargetGroupArn
                    }
                }
            });
        return createLbResponse.LoadBalancers[0];
    }

    /// <summary>
```

```
    /// Verify this computer can successfully send a GET request to the
    /// load balancer endpoint.
    /// </summary>
    /// <param name="endpoint">The endpoint to check.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> VerifyLoadBalancerEndpoint(string endpoint)
    {
        var success = false;
        var retries = 3;
        while (!success && retries > 0)
        {
            try
            {
                var endpointResponse = await _httpClient.GetAsync($"http://
{endpoint}");
                Console.WriteLine($"Response: {endpointResponse.StatusCode}.");

                if (endpointResponse.IsSuccessStatusCode)
                {
                    success = true;
                }
                else
                {
                    retries = 0;
                }
            }
            catch (HttpRequestException)
            {
                Console.WriteLine("Connection error, retrying...");
                retries--;
                Thread.Sleep(10000);
            }
        }

        return success;
    }

    /// <summary>
    /// Delete a load balancer by its specified name.
    /// </summary>
    /// <param name="name">The name of the load balancer to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteLoadBalancerByName(string name)
    {
```

```
        try
        {
            var describeLoadBalancerResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                    new DescribeLoadBalancersRequest()
                    {
                        Names = new List<string>() { name }
                    });
            var lbArn =
describeLoadBalancerResponse.LoadBalancers[0].LoadBalancerArn;
            await _amazonElasticLoadBalancingV2.DeleteLoadBalancerAsync(
                new DeleteLoadBalancerRequest()
                {
                    LoadBalancerArn = lbArn
                }
            );
        }
        catch (LoadBalancerNotFoundException)
        {
            Console.WriteLine($"Load balancer {name} not found.");
        }
    }

    /// <summary>
    /// Delete a TargetGroup by its specified name.
    /// </summary>
    /// <param name="groupName">Name of the group to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteTargetGroupByName(string groupName)
    {
        var done = false;
        while (!done)
        {
            try
            {
                var groupResponse =
                    await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                        new DescribeTargetGroupsRequest()
                        {
                            Names = new List<string>() { groupName }
                        });

                var targetArn = groupResponse.TargetGroups[0].TargetGroupArn;
                await _amazonElasticLoadBalancingV2.DeleteTargetGroupAsync(
```

```
                    new DeleteTargetGroupRequest() { TargetGroupArn = targetArn });
                Console.WriteLine($"Deleted load balancing target group
 {groupName}.");
                done = true;
            }
            catch (TargetGroupNotFoundException)
            {
                Console.WriteLine(
                    $"Target group {groupName} not found, could not delete.");
                done = true;
            }
            catch (ResourceInUseException)
            {
                Console.WriteLine("Target group not yet released, waiting...");
                Thread.Sleep(10000);
            }
        }
    }
}
```

DynamoDB を使用してレコメンデーションサービスをシミュレートするクラスを作成します。

```
/// <summary>
/// Encapsulates a DynamoDB table to use as a service that recommends books, movies,
 and songs.
/// </summary>
public class Recommendations
{
    private readonly IAmazonDynamoDB _amazonDynamoDb;
    private readonly DynamoDBContext _context;
    private readonly string _tableName;

    public string TableName => _tableName;

    /// <summary>
    /// Constructor for the Recommendations service.
    /// </summary>
    /// <param name="amazonDynamoDb">The injected DynamoDb client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public Recommendations(IAmazonDynamoDB amazonDynamoDb, IConfiguration
 configuration)
    {
```

```
        _amazonDynamoDb = amazonDynamoDb;
        _context = new DynamoDBContext(_amazonDynamoDb);
        _tableName = configuration["databaseName"]!;
    }

    /// <summary>
    /// Create the DynamoDb table with a specified name.
    /// </summary>
    /// <param name="tableName">The name for the table.</param>
    /// <returns>True when ready.</returns>
    public async Task<bool> CreateDatabaseWithName(string tableName)
    {
        try
        {
            Console.Write($"Creating table {tableName}...");
            var createRequest = new CreateTableRequest()
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                    {
                        new AttributeDefinition()
                        {
                            AttributeName = "MediaType",
                            AttributeType = ScalarAttributeType.S
                        },
                        new AttributeDefinition()
                        {
                            AttributeName = "ItemId",
                            AttributeType = ScalarAttributeType.N
                        }
                    },
                KeySchema = new List<KeySchemaElement>()
                    {
                        new KeySchemaElement()
                        {
                            AttributeName = "MediaType",
                            KeyType = KeyType.HASH
                        },
                        new KeySchemaElement()
                        {
                            AttributeName = "ItemId",
                            KeyType = KeyType.RANGE
                        }
                    },
```

```
                ProvisionedThroughput = new ProvisionedThroughput()
                {
                    ReadCapacityUnits = 5,
                    WriteCapacityUnits = 5
                }
            };
            await _amazonDynamoDb.CreateTableAsync(createRequest);

            // Wait until the table is ACTIVE and then report success.
            Console.Write("\nWaiting for table to become active...");

            var request = new DescribeTableRequest
            {
                TableName = tableName
            };

            TableStatus status;
            do
            {
                Thread.Sleep(2000);

                var describeTableResponse = await
 _amazonDynamoDb.DescribeTableAsync(request);
                status = describeTableResponse.Table.TableStatus;

                Console.Write(".");
            }
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException)
        {
            Console.WriteLine($"Table {tableName} already exists.");
            return false;
        }
    }

    /// <summary>
    /// Populate the database table with data from a specified path.
    /// </summary>
    /// <param name="databaseTableName">The name of the table.</param>
    /// <param name="recommendationsPath">The path of the recommendations data.</
param>
```

```
    /// <returns>Async task.</returns>
    public async Task PopulateDatabase(string databaseTableName, string
 recommendationsPath)
    {
        var recommendationsText = await File.ReadAllTextAsync(recommendationsPath);
        var records =
            JsonSerializer.Deserialize<RecommendationModel[]>(recommendationsText);
        var batchWrite = _context.CreateBatchWrite<RecommendationModel>();

        foreach (var record in records!)
        {
            batchWrite.AddPutItem(record);
        }

        await batchWrite.ExecuteAsync();
    }

    /// <summary>
    /// Delete the recommendation table by name.
    /// </summary>
    /// <param name="tableName">The name of the recommendation table.</param>
    /// <returns>Async task.</returns>
    public async Task DestroyDatabaseByName(string tableName)
    {
        try
        {
            await _amazonDynamoDb.DeleteTableAsync(
                new DeleteTableRequest() { TableName = tableName });
            Console.WriteLine($"Table {tableName} was deleted.");
        }
        catch (ResourceNotFoundException)
        {
            Console.WriteLine($"Table {tableName} not found");
        }
    }
}
```

Systems Manager のアクションをラップするクラスを作成します。

```
/// <summary>
/// Encapsulates Systems Manager parameter operations. This example uses these
 parameters
```

```
/// to drive the demonstration of resilient architecture, such as failure of a
 dependency or
/// how the service responds to a health check.
/// </summary>
public class SmParameterWrapper
{
    private readonly IAmazonSimpleSystemsManagement _amazonSimpleSystemsManagement;

    private readonly string _tableParameter = "doc-example-resilient-architecture-
table";
    private readonly string _failureResponseParameter = "doc-example-resilient-
architecture-failure-response";
    private readonly string _healthCheckParameter = "doc-example-resilient-
architecture-health-check";
    private readonly string _tableName = "";

    public string TableParameter => _tableParameter;
    public string TableName => _tableName;
    public string HealthCheckParameter => _healthCheckParameter;
    public string FailureResponseParameter => _failureResponseParameter;

    /// <summary>
    /// Constructor for the SmParameterWrapper.
    /// </summary>
    /// <param name="amazonSimpleSystemsManagement">The injected Simple Systems
 Management client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public SmParameterWrapper(IAmazonSimpleSystemsManagement
 amazonSimpleSystemsManagement, IConfiguration configuration)
    {
        _amazonSimpleSystemsManagement = amazonSimpleSystemsManagement;
        _tableName = configuration["databaseName"]!;
    }

    /// <summary>
    /// Reset the Systems Manager parameters to starting values for the demo.
    /// </summary>
    /// <returns>Async task.</returns>
    public async Task Reset()
    {
        await this.PutParameterByName(_tableParameter, _tableName);
        await this.PutParameterByName(_failureResponseParameter, "none");
        await this.PutParameterByName(_healthCheckParameter, "shallow");
    }
```

```
    /// <summary>
    /// Set the value of a named Systems Manager parameter.
    /// </summary>
    /// <param name="name">The name of the parameter.</param>
    /// <param name="value">The value to set.</param>
    /// <returns>Async task.</returns>
    public async Task PutParameterByName(string name, string value)
    {
        await _amazonSimpleSystemsManagement.PutParameterAsync(
            new PutParameterRequest() { Name = name, Value = value, Overwrite =
 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。
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer
  - CreateTargetGroup
  - DeleteAutoScalingGroup
  - DeleteInstanceProfile
  - DeleteLaunchTemplate
  - DeleteLoadBalancer
  - DeleteTargetGroup
  - DescribeAutoScalingGroups
  - DescribeAvailabilityZones
  - DescribeIamInstanceProfileAssociations
  - DescribeInstances
  - DescribeLoadBalancers

- DescribeSubnets
- DescribeTargetGroups
- DescribeTargetHealth
- DescribeVpcs
- RebootInstances
- ReplaceIamInstanceProfileAssociation
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

インスタンスを開始

次のコードサンプルは、以下の操作方法を示しています。

- キーペアとセキュリティグループを作成します。
- Amazon マシンイメージ (AMI)と互換性のあるインスタンスタイプを選択し、インスタンスを作成します。
- インスタンスを停止し、再起動します。
- Elastic IP アドレスをインスタンスに関連付ける。
- SSH を使用してインスタンスに接続し、リソースをクリーンアップします。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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でシナリオを実行します。

```
/// <summary>
/// Show Amazon Elastic Compute Cloud (Amazon EC2) Basics actions.
/// </summary>
public class EC2Basics
{
    /// <summary>
```

```
    /// Perform the actions defined for the Amazon EC2 Basics scenario.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A Task object.</returns>
    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon EC2 and Amazon Simple Systems
        // Management Service.
        using var host =
Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonEC2>()
                    .AddAWSService<IAmazonSimpleSystemsManagement>()
                    .AddTransient<EC2Wrapper>()
                    .AddTransient<SsmWrapper>()
            )
            .Build();

        // Now the client is available for injection.
        var ec2Client = host.Services.GetRequiredService<IAmazonEC2>();
        var ec2Methods = new EC2Wrapper(ec2Client);

        var ssmClient =
host.Services.GetRequiredService<IAmazonSimpleSystemsManagement>();
        var ssmMethods = new SsmWrapper(ssmClient);
        var uiMethods = new UiMethods();

        var uniqueName = Guid.NewGuid().ToString();
        var keyPairName = "mvp-example-key-pair" + uniqueName;
        var groupName = "ec2-scenario-group" + uniqueName;
        var groupDescription = "A security group created for the EC2 Basics
scenario.";

        // Start the scenario.
        uiMethods.DisplayOverview();
        uiMethods.PressEnter();

        // Create the key pair.
        uiMethods.DisplayTitle("Create RSA key pair");
        Console.Write("Let's create an RSA key pair that you can be use to ");
        Console.WriteLine("securely connect to your EC2 instance.");
        var keyPair = await ec2Methods.CreateKeyPair(keyPairName);

        // Save key pair information to a temporary file.
```

```
        var tempFileName = ec2Methods.SaveKeyPair(keyPair);

        Console.WriteLine($"Created the key pair: {keyPair.KeyName} and saved it to:
{tempFileName}");
        string? answer;
        do
        {
            Console.Write("Would you like to list your existing key pairs? ");
            answer = Console.ReadLine();
        } while (answer!.ToLower() != "y" && answer.ToLower() != "n");

        if (answer == "y")
        {
            // List existing key pairs.
            uiMethods.DisplayTitle("Existing key pairs");

            // Passing an empty string to the DescribeKeyPairs method will return
            // a list of all existing key pairs.
            var keyPairs = await ec2Methods.DescribeKeyPairs("");
            keyPairs.ForEach(kp =>
            {
                Console.WriteLine($"{kp.KeyName} created at: {kp.CreateTime}
Fingerprint: {kp.KeyFingerprint}");
            });
        }
        uiMethods.PressEnter();

        // Create the security group.
        Console.WriteLine("Let's create a security group to manage access to your
instance.");
        var secGroupId = await ec2Methods.CreateSecurityGroup(groupName,
groupDescription);
        Console.WriteLine("Let's add rules to allow all HTTP and HTTPS inbound
traffic and to allow SSH only from your current IP address.");

        uiMethods.DisplayTitle("Security group information");
        var secGroups = await ec2Methods.DescribeSecurityGroups(secGroupId);

        Console.WriteLine($"Created security group {groupName} in your default
VPC.");
        secGroups.ForEach(group =>
        {
            ec2Methods.DisplaySecurityGroupInfoAsync(group);
        });
```

```
        uiMethods.PressEnter();

        Console.WriteLine("Now we'll authorize the security group we just created so
 that it can");
        Console.WriteLine("access the EC2 instances you create.");
        var success = await ec2Methods.AuthorizeSecurityGroupIngress(groupName);

        secGroups = await ec2Methods.DescribeSecurityGroups(secGroupId);
        Console.WriteLine($"Now let's look at the permissions again.");
        secGroups.ForEach(group =>
        {
            ec2Methods.DisplaySecurityGroupInfoAsync(group);
        });
        uiMethods.PressEnter();

        // Get list of available Amazon Linux 2 Amazon Machine Images (AMIs).
        var parameters = await ssmMethods.GetParametersByPath("/aws/service/ami-
amazon-linux-latest");

        List<string> imageIds = parameters.Select(param => param.Value).ToList();

        var images = await ec2Methods.DescribeImages(imageIds);

        var i = 1;
        images.ForEach(image =>
        {
            Console.WriteLine($"\t{i++}\t{image.Description}");
        });

        int choice;
        bool validNumber = false;

        do
        {
            Console.Write("Please select an image: ");
            var selImage = Console.ReadLine();
            validNumber = int.TryParse(selImage, out choice);
        } while (!validNumber);

        var selectedImage = images[choice - 1];

        // Display available instance types.
        uiMethods.DisplayTitle("Instance Types");
```

```
        var instanceTypes = await
 ec2Methods.DescribeInstanceTypes(selectedImage.Architecture);

        i = 1;
        instanceTypes.ForEach(instanceType =>
        {
            Console.WriteLine($"\t{i++}\t{instanceType.InstanceType}");
        });

        do
        {
            Console.Write("Please select an instance type: ");
            var selImage = Console.ReadLine();
            validNumber = int.TryParse(selImage, out choice);
        } while (!validNumber);

        var selectedInstanceType = instanceTypes[choice - 1].InstanceType;

        // Create an EC2 instance.
        uiMethods.DisplayTitle("Creating an EC2 Instance");
        var instanceId = await ec2Methods.RunInstances(selectedImage.ImageId,
 selectedInstanceType, keyPairName, secGroupId);
        Console.Write("Waiting for the instance to start.");
        var isRunning = false;
        do
        {
            isRunning = await ec2Methods.WaitForInstanceState(instanceId,
 InstanceStateName.Running);
        } while (!isRunning);

        uiMethods.PressEnter();

        var instance = await ec2Methods.DescribeInstance(instanceId);
        uiMethods.DisplayTitle("New Instance Information");
        ec2Methods.DisplayInstanceInformation(instance);

        Console.WriteLine("\nYou can use SSH to connect to your instance. For
 example:");
        Console.WriteLine($"\tssh -i {tempFileName} ec2-
user@{instance.PublicIpAddress}");

        uiMethods.PressEnter();
```

```
        Console.WriteLine("Now we'll stop the instance and then start it again to
 see what's changed.");

        await ec2Methods.StopInstances(instanceId);
        var hasStopped = false;
        do
        {
            hasStopped = await ec2Methods.WaitForInstanceState(instanceId,
 InstanceStateName.Stopped);
        } while (!hasStopped);

        Console.WriteLine("\nThe instance has stopped.");

        Console.WriteLine("Now let's start it up again.");
        await ec2Methods.StartInstances(instanceId);
        Console.Write("Waiting for instance to start. ");

        isRunning = false;
        do
        {
            isRunning = await ec2Methods.WaitForInstanceState(instanceId,
 InstanceStateName.Running);
        } while (!isRunning);

        Console.WriteLine("\nLet's see what changed.");

        instance = await ec2Methods.DescribeInstance(instanceId);
        uiMethods.DisplayTitle("New Instance Information");
        ec2Methods.DisplayInstanceInformation(instance);

        Console.WriteLine("\nNotice the change in the SSH information:");
        Console.WriteLine($"\tssh -i {tempFileName} ec2-
user@{instance.PublicIpAddress}");

        uiMethods.PressEnter();

        Console.WriteLine("Now we will stop the instance again. Then we will create
 and associate an");
        Console.WriteLine("Elastic IP address to use with our instance.");

        await ec2Methods.StopInstances(instanceId);
        hasStopped = false;
        do
        {
```

```
            hasStopped = await ec2Methods.WaitForInstanceState(instanceId,
 InstanceStateName.Stopped);
        } while (!hasStopped);

        Console.WriteLine("\nThe instance has stopped.");
        uiMethods.PressEnter();

        uiMethods.DisplayTitle("Allocate Elastic IP address");
        Console.WriteLine("You can allocate an Elastic IP address and associate
 it with your instance\nto keep a consistent IP address even when your instance
 restarts.");
        var allocationId = await ec2Methods.AllocateAddress();
        Console.WriteLine("Now we will associate the Elastic IP address with our
 instance.");
        var associationId = await ec2Methods.AssociateAddress(allocationId,
 instanceId);

        // Start the instance again.
        Console.WriteLine("Now let's start the instance again.");
        await ec2Methods.StartInstances(instanceId);
        Console.Write("Waiting for instance to start. ");

        isRunning = false;
        do
        {
            isRunning = await ec2Methods.WaitForInstanceState(instanceId,
 InstanceStateName.Running);
        } while (!isRunning);

        Console.WriteLine("\nLet's see what changed.");

        instance = await ec2Methods.DescribeInstance(instanceId);
        uiMethods.DisplayTitle("Instance information");
        ec2Methods.DisplayInstanceInformation(instance);

        Console.WriteLine("\nHere is the SSH information:");
        Console.WriteLine($"\tssh -i {tempFileName} ec2-
user@{instance.PublicIpAddress}");

        Console.WriteLine("Let's stop and start the instance again.");
        uiMethods.PressEnter();

        await ec2Methods.StopInstances(instanceId);
```

```
        hasStopped = false;
        do
        {
            hasStopped = await ec2Methods.WaitForInstanceState(instanceId,
InstanceStateName.Stopped);
        } while (!hasStopped);

        Console.WriteLine("\nThe instance has stopped.");

        Console.WriteLine("Now let's start it up again.");
        await ec2Methods.StartInstances(instanceId);
        Console.Write("Waiting for instance to start. ");

        isRunning = false;
        do
        {
            isRunning = await ec2Methods.WaitForInstanceState(instanceId,
InstanceStateName.Running);
        } while (!isRunning);

        instance = await ec2Methods.DescribeInstance(instanceId);
        uiMethods.DisplayTitle("New Instance Information");
        ec2Methods.DisplayInstanceInformation(instance);
        Console.WriteLine("Note that the IP address did not change this time.");
        uiMethods.PressEnter();

        uiMethods.DisplayTitle("Clean up resources");

        Console.WriteLine("Now let's clean up the resources we created.");

        // Terminate the instance.
        Console.WriteLine("Terminating the instance we created.");
        var stateChange = await ec2Methods.TerminateInstances(instanceId);

        // Wait for the instance state to be terminated.
        var hasTerminated = false;
        do
        {
            hasTerminated = await ec2Methods.WaitForInstanceState(instanceId,
InstanceStateName.Terminated);
        } while (!hasTerminated);

        Console.WriteLine($"\nThe instance {instanceId} has been terminated.");
```

```
        Console.WriteLine("Now we can disassociate the Elastic IP address and
 release it.");

        // Disassociate the Elastic IP address.
        var disassociated = ec2Methods.DisassociateIp(associationId);

        // Delete the Elastic IP address.
        var released = ec2Methods.ReleaseAddress(allocationId);

        // Delete the security group.
        Console.WriteLine($"Deleting the Security Group: {groupName}.");
        success = await ec2Methods.DeleteSecurityGroup(secGroupId);
        if (success)
        {
            Console.WriteLine($"Successfully deleted {groupName}.");
        }

        // Delete the RSA key pair.
        Console.WriteLine($"Deleting the key pair: {keyPairName}");
        await ec2Methods.DeleteKeyPair(keyPairName);
        Console.WriteLine("Deleting the temporary file with the key information.");
        ec2Methods.DeleteTempFile(tempFileName);
        uiMethods.PressEnter();

        uiMethods.DisplayTitle("EC2 Basics Scenario completed.");
        uiMethods.PressEnter();
    }
}
```

EC2 アクションをラップするクラスを定義します。

```
/// <summary>
/// Methods of this class perform Amazon Elastic Compute Cloud (Amazon EC2).
/// </summary>
public class EC2Wrapper
{
    private readonly IAmazonEC2 _amazonEC2;

    public EC2Wrapper(IAmazonEC2 amazonService)
    {
        _amazonEC2 = amazonService;
    }
```

```
    /// <summary>
    /// Allocate an Elastic IP address.
    /// </summary>
    /// <returns>The allocation Id of the allocated address.</returns>
    public async Task<string> AllocateAddress()
    {
        var request = new AllocateAddressRequest();

        var response = await _amazonEC2.AllocateAddressAsync(request);
        return response.AllocationId;
    }

    /// <summary>
    /// Associate an Elastic IP address to an EC2 instance.
    /// </summary>
    /// <param name="allocationId">The allocation Id of an Elastic IP address.</
param>
    /// <param name="instanceId">The instance Id of the EC2 instance to
    /// associate the address with.</param>
    /// <returns>The association Id that represents
    /// the association of the Elastic IP address with an instance.</returns>
    public async Task<string> AssociateAddress(string allocationId, string
 instanceId)
    {
        var request = new AssociateAddressRequest
        {
            AllocationId = allocationId,
            InstanceId = instanceId
        };

        var response = await _amazonEC2.AssociateAddressAsync(request);
        return response.AssociationId;
    }

    /// <summary>
    /// Authorize the local computer ingress to EC2 instances associated
    /// with the virtual private cloud (VPC) security group.
    /// </summary>
    /// <param name="groupName">The name of the security group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AuthorizeSecurityGroupIngress(string groupName)
    {
        // Get the IP address for the local computer.
```

```
        var ipAddress = await GetIpAddress();
        Console.WriteLine($"Your IP address is: {ipAddress}");
        var ipRanges = new List<IpRange> { new IpRange { CidrIp =
 $"{ipAddress}/32" } };
        var permission = new IpPermission
        {
            Ipv4Ranges = ipRanges,
            IpProtocol = "tcp",
            FromPort = 22,
            ToPort = 22
        };
        var permissions = new List<IpPermission> { permission };
        var response = await _amazonEC2.AuthorizeSecurityGroupIngressAsync(
            new AuthorizeSecurityGroupIngressRequest(groupName, permissions));
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Authorize the local computer for ingress to
    /// the Amazon EC2 SecurityGroup.
    /// </summary>
    /// <returns>The IPv4 address of the computer running the scenario.</returns>
    private static async Task<string> GetIpAddress()
    {
        var httpClient = new HttpClient();
        var ipString = await httpClient.GetStringAsync("https://
checkip.amazonaws.com");

        // The IP address is returned with a new line
        // character on the end. Trim off the whitespace and
        // return the value to the caller.
        return ipString.Trim();
    }

    /// <summary>
    /// Create an Amazon EC2 key pair.
    /// </summary>
    /// <param name="keyPairName">The name for the new key pair.</param>
    /// <returns>The Amazon EC2 key pair created.</returns>
    public async Task<KeyPair?> CreateKeyPair(string keyPairName)
    {
        var request = new CreateKeyPairRequest
        {
            KeyName = keyPairName,
```

```
        };

        var response = await _amazonEC2.CreateKeyPairAsync(request);

        if (response.HttpStatusCode == HttpStatusCode.OK)
        {
            var kp = response.KeyPair;
            return kp;
        }
        else
        {
            Console.WriteLine("Could not create key pair.");
            return null;
        }
    }

    /// <summary>
    /// Save KeyPair information to a temporary file.
    /// </summary>
    /// <param name="keyPair">The name of the key pair.</param>
    /// <returns>The full path to the temporary file.</returns>
    public string SaveKeyPair(KeyPair keyPair)
    {
        var tempPath = Path.GetTempPath();
        var tempFileName = $"{tempPath}\\{Path.GetRandomFileName()}";
        var pemFileName = Path.ChangeExtension(tempFileName, "pem");

        // Save the key pair to a file in a temporary folder.
        using var stream = new FileStream(pemFileName, FileMode.Create);
        using var writer = new StreamWriter(stream);
        writer.WriteLine(keyPair.KeyMaterial);

        return pemFileName;
    }

    /// <summary>
    /// Create an Amazon EC2 security group.
    /// </summary>
    /// <param name="groupName">The name for the new security group.</param>
    /// <param name="groupDescription">A description of the new security group.</
param>
    /// <returns>The group Id of the new security group.</returns>
    public async Task<string> CreateSecurityGroup(string groupName, string
 groupDescription)
```

```
    {
        var response = await _amazonEC2.CreateSecurityGroupAsync(
            new CreateSecurityGroupRequest(groupName, groupDescription));

        return response.GroupId;
    }


    /// <summary>
    /// Create a new Amazon EC2 VPC.
    /// </summary>
    /// <param name="cidrBlock">The CIDR block for the new security group.</param>
    /// <returns>The VPC Id of the new VPC.</returns>
    public async Task<string?> CreateVPC(string cidrBlock)
    {

        try
        {
            var response = await _amazonEC2.CreateVpcAsync(new CreateVpcRequest
            {
                CidrBlock = cidrBlock,
            });

            Vpc vpc = response.Vpc;
            Console.WriteLine($"Created VPC with ID: {vpc.VpcId}.");
            return vpc.VpcId;
        }
        catch (AmazonEC2Exception ex)
        {
            Console.WriteLine($"Couldn't create VPC because: {ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Delete an Amazon EC2 key pair.
    /// </summary>
    /// <param name="keyPairName">The name of the key pair to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteKeyPair(string keyPairName)
    {
        try
        {
```

```
            await _amazonEC2.DeleteKeyPairAsync(new
DeleteKeyPairRequest(keyPairName)).ConfigureAwait(false);
            return true;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't delete the key pair because:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Delete the temporary file where the key pair information was saved.
    /// </summary>
    /// <param name="tempFileName">The path to the temporary file.</param>
    public void DeleteTempFile(string tempFileName)
    {
        if (File.Exists(tempFileName))
        {
            File.Delete(tempFileName);
        }
    }

    /// <summary>
    /// Delete an Amazon EC2 security group.
    /// </summary>
    /// <param name="groupName">The name of the group to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteSecurityGroup(string groupId)
    {
        var response = await _amazonEC2.DeleteSecurityGroupAsync(new
DeleteSecurityGroupRequest { GroupId = groupId });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete an Amazon EC2 VPC.
    /// </summary>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteVpc(string vpcId)
    {
        var request = new DeleteVpcRequest
        {
```

```
            VpcId = vpcId,
        };

        var response = await _amazonEC2.DeleteVpcAsync(request);

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Get information about existing Amazon EC2 images.
    /// </summary>
    /// <returns>A list of image information.</returns>
    public async Task<List<Image>> DescribeImages(List<string>? imageIds)
    {
        var request = new DescribeImagesRequest();
        if (imageIds is not null)
        {
            // If the imageIds list is not null, add the list
            // to the request object.
            request.ImageIds = imageIds;
        }

        var response = await _amazonEC2.DescribeImagesAsync(request);
        return response.Images;
    }

    /// <summary>
    /// Display the information returned by DescribeImages.
    /// </summary>
    /// <param name="images">The list of image information to display.</param>
    public void DisplayImageInfo(List<Image> images)
    {
        images.ForEach(image =>
        {
            Console.WriteLine($"{image.Name} Created on: {image.CreationDate}");
        });

    }

    /// <summary>
    /// Get information about an Amazon EC2 instance.
    /// </summary>
    /// <param name="instanceId">The instance Id of the EC2 instance.</param>
    /// <returns>An EC2 instance.</returns>
```

```
    public async Task<Instance> DescribeInstance(string instanceId)
    {
        var response = await _amazonEC2.DescribeInstancesAsync(
            new DescribeInstancesRequest { InstanceIds = new List<string>
{ instanceId } });
        return response.Reservations[0].Instances[0];
    }

    /// <summary>
    /// Display EC2 instance information.
    /// </summary>
    /// <param name="instance">The instance Id of the EC2 instance.</param>
    public void DisplayInstanceInformation(Instance instance)
    {
        Console.WriteLine($"ID: {instance.InstanceId}");
        Console.WriteLine($"Image ID: {instance.ImageId}");
        Console.WriteLine($"{instance.InstanceType}");
        Console.WriteLine($"Key Name: {instance.KeyName}");
        Console.WriteLine($"VPC ID: {instance.VpcId}");
        Console.WriteLine($"Public IP: {instance.PublicIpAddress}");
        Console.WriteLine($"State: {instance.State.Name}");
    }

    /// <summary>
    /// Get information about existing EC2 images.
    /// </summary>
    /// <returns>Async task.</returns>
    public async Task DescribeInstances()
    {
        // List all EC2 instances.
        await GetInstanceDescriptions();

        string tagName = "IncludeInList";
        string tagValue = "Yes";
        await GetInstanceDescriptionsFiltered(tagName, tagValue);
    }

    /// <summary>
    /// Get information for all existing Amazon EC2 instances.
    /// </summary>
    /// <returns>Async task.</returns>
    public async Task GetInstanceDescriptions()
    {
        Console.WriteLine("Showing all instances:");
```

```
        var paginator = _amazonEC2.Paginators.DescribeInstances(new
DescribeInstancesRequest());

        await foreach (var response in paginator.Responses)
        {
            foreach (var reservation in response.Reservations)
            {
                foreach (var instance in reservation.Instances)
                {
                    Console.Write($"Instance ID: {instance.InstanceId}");
                    Console.WriteLine($"\tCurrent State: {instance.State.Name}");
                }
            }
        }
    }

    /// <summary>
    /// Get information about EC2 instances filtered by a tag name and value.
    /// </summary>
    /// <param name="tagName">The name of the tag to filter on.</param>
    /// <param name="tagValue">The value of the tag to look for.</param>
    /// <returns>Async task.</returns>
    public async Task GetInstanceDescriptionsFiltered(string tagName, string
tagValue)
    {
        // This tag filters the results of the instance list.
        var filters = new List<Filter>
        {
            new Filter
            {
                Name = $"tag:{tagName}",
                Values = new List<string>
                {
                    tagValue,
                },
            },
        };
        var request = new DescribeInstancesRequest
        {
            Filters = filters,
        };

        Console.WriteLine("\nShowing instances with tag: \"IncludeInList\" set to
\"Yes\".");
```

```
        var paginator = _amazonEC2.Paginators.DescribeInstances(request);

        await foreach (var response in paginator.Responses)
        {
            foreach (var reservation in response.Reservations)
            {
                foreach (var instance in reservation.Instances)
                {
                    Console.Write($"Instance ID: {instance.InstanceId} ");
                    Console.WriteLine($"\tCurrent State: {instance.State.Name}");
                }
            }
        }
    }

    /// <summary>
    /// Describe the instance types available.
    /// </summary>
    /// <returns>A list of instance type information.</returns>
    public async Task<List<InstanceTypeInfo>>
DescribeInstanceTypes(ArchitectureValues architecture)
    {
        var request = new DescribeInstanceTypesRequest();

        var filters = new List<Filter>
            { new Filter("processor-info.supported-architecture", new List<string>
{ architecture.ToString() }) };
        filters.Add(new Filter("instance-type", new() { "*.micro", "*.small" }));

        request.Filters = filters;
        var instanceTypes = new List<InstanceTypeInfo>();

        var paginator = _amazonEC2.Paginators.DescribeInstanceTypes(request);
        await foreach (var instanceType in paginator.InstanceTypes)
        {
            instanceTypes.Add(instanceType);
        }
        return instanceTypes;
    }

    /// <summary>
    /// Display the instance type information returned by
DescribeInstanceTypesAsync.
    /// </summary>
```

```
    /// <param name="instanceTypes">The list of instance type information.</param>
    public void DisplayInstanceTypeInfo(List<InstanceTypeInfo> instanceTypes)
    {
        instanceTypes.ForEach(type =>
        {
            Console.WriteLine($"{type.InstanceType}\t{type.MemoryInfo}");
        });
    }

    /// <summary>
    /// Get information about an Amazon EC2 key pair.
    /// </summary>
    /// <param name="keyPairName">The name of the key pair.</param>
    /// <returns>A list of key pair information.</returns>
    public async Task<List<KeyPairInfo>> DescribeKeyPairs(string keyPairName)
    {
        var request = new DescribeKeyPairsRequest();
        if (!string.IsNullOrEmpty(keyPairName))
        {
            request = new DescribeKeyPairsRequest
            {
                KeyNames = new List<string> { keyPairName }
            };
        }
        var response = await _amazonEC2.DescribeKeyPairsAsync(request);
        return response.KeyPairs.ToList();
    }


    /// <summary>
    /// Retrieve information for an Amazon EC2 security group.
    /// </summary>
    /// <param name="groupId">The Id of the Amazon EC2 security group.</param>
    /// <returns>A list of security group information.</returns>
    public async Task<List<SecurityGroup>> DescribeSecurityGroups(string groupId)
    {
        var request = new DescribeSecurityGroupsRequest();
        var groupIds = new List<string> { groupId };
        request.GroupIds = groupIds;

        var response = await _amazonEC2.DescribeSecurityGroupsAsync(request);
        return response.SecurityGroups;
    }
```

```
    /// <summary>
    /// Display the information returned by the call to
    /// DescribeSecurityGroupsAsync.
    /// </summary>
    /// <param name="securityGroup">A list of security group information.</param>
    public void DisplaySecurityGroupInfoAsync(SecurityGroup securityGroup)
    {
        Console.WriteLine($"{securityGroup.GroupName}");
        Console.WriteLine("Ingress permissions:");
        securityGroup.IpPermissions.ForEach(permission =>
        {
            Console.WriteLine($"\tFromPort: {permission.FromPort}");
            Console.WriteLine($"\tIpProtocol: {permission.IpProtocol}");

            Console.Write($"\tIpv4Ranges: ");
            permission.Ipv4Ranges.ForEach(range => { Console.Write($"{range.CidrIp}
"); });

            Console.WriteLine($"\n\tIpv6Ranges:");
            permission.Ipv6Ranges.ForEach(range =>
{ Console.Write($"{range.CidrIpv6} "); });

            Console.Write($"\n\tPrefixListIds: ");
            permission.PrefixListIds.ForEach(id => Console.Write($"{id.Id} "));

            Console.WriteLine($"\n\tTo Port: {permission.ToPort}");
        });
        Console.WriteLine("Egress permissions:");
        securityGroup.IpPermissionsEgress.ForEach(permission =>
        {
            Console.WriteLine($"\tFromPort: {permission.FromPort}");
            Console.WriteLine($"\tIpProtocol: {permission.IpProtocol}");

            Console.Write($"\tIpv4Ranges: ");
            permission.Ipv4Ranges.ForEach(range => { Console.Write($"{range.CidrIp}
"); });

            Console.WriteLine($"\n\tIpv6Ranges:");
            permission.Ipv6Ranges.ForEach(range =>
{ Console.Write($"{range.CidrIpv6} "); });

            Console.Write($"\n\tPrefixListIds: ");
            permission.PrefixListIds.ForEach(id => Console.Write($"{id.Id} "));
```

```
            Console.WriteLine($"\n\tTo Port: {permission.ToPort}");
        });
    }


    /// <summary>
    /// Disassociate an Elastic IP address from an EC2 instance.
    /// </summary>
    /// <param name="associationId">The association Id.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DisassociateIp(string associationId)
    {
        var response = await _amazonEC2.DisassociateAddressAsync(
            new DisassociateAddressRequest { AssociationId = associationId });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Retrieve a list of available Amazon Linux images.
    /// </summary>
    /// <returns>A list of image information.</returns>
    public async Task<List<Image>> GetEC2AmiList()
    {
        var filter = new Filter { Name = "architecture", Values = new List<string>
{ "x86_64" } };
        var filters = new List<Filter> { filter };
        var response = await _amazonEC2.DescribeImagesAsync(new
DescribeImagesRequest { Filters = filters });
        return response.Images;
    }

    /// <summary>
    /// Reboot EC2 instances.
    /// </summary>
    /// <param name="ec2InstanceId">The instance Id of the instances that will be
rebooted.</param>
    /// <returns>Async task.</returns>
    public async Task RebootInstances(string ec2InstanceId)
    {
        var request = new RebootInstancesRequest
        {
            InstanceIds = new List<string> { ec2InstanceId },
        };
```

```
        var response = await _amazonEC2.RebootInstancesAsync(request);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine("Instances successfully rebooted.");
        }
        else
        {
            Console.WriteLine("Could not reboot one or more instances.");
        }
    }

    /// <summary>
    /// Release an Elastic IP address.
    /// </summary>
    /// <param name="allocationId">The allocation Id of the Elastic IP address.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> ReleaseAddress(string allocationId)
    {
        var request = new ReleaseAddressRequest
        {
            AllocationId = allocationId
        };

        var response = await _amazonEC2.ReleaseAddressAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Create and run an EC2 instance.
    /// </summary>
    /// <param name="ImageId">The image Id of the image used as a basis for the
    /// EC2 instance.</param>
    /// <param name="instanceType">The instance type of the EC2 instance to
 create.</param>
    /// <param name="keyName">The name of the key pair to associate with the
    /// instance.</param>
    /// <param name="groupId">The Id of the Amazon EC2 security group that will be
    /// allowed to interact with the new EC2 instance.</param>
    /// <returns>The instance Id of the new EC2 instance.</returns>
    public async Task<string> RunInstances(string imageId, string instanceType,
 string keyName, string groupId)
    {
        var request = new RunInstancesRequest
```

```
        {
            ImageId = imageId,
            InstanceType = instanceType,
            KeyName = keyName,
            MinCount = 1,
            MaxCount = 1,
            SecurityGroupIds = new List<string> { groupId }
        };
        var response = await _amazonEC2.RunInstancesAsync(request);
        return response.Reservation.Instances[0].InstanceId;
    }


    /// <summary>
    /// Start an EC2 instance.
    /// </summary>
    /// <param name="ec2InstanceId">The instance Id of the Amazon EC2 instance
    /// to start.</param>
    /// <returns>Async task.</returns>
    public async Task StartInstances(string ec2InstanceId)
    {
        var request = new StartInstancesRequest
        {
            InstanceIds = new List<string> { ec2InstanceId },
        };

        var response = await _amazonEC2.StartInstancesAsync(request);

        if (response.StartingInstances.Count > 0)
        {
            var instances = response.StartingInstances;
            instances.ForEach(i =>
            {
                Console.WriteLine($"Successfully started the EC2 instance with
instance ID: {i.InstanceId}.");
            });
        }
    }

    /// <summary>
    /// Stop an EC2 instance.
    /// </summary>
    /// <param name="ec2InstanceId">The instance Id of the EC2 instance to
    /// stop.</param>
```

```
    /// <returns>Async task.</returns>
    public async Task StopInstances(string ec2InstanceId)
    {
        // In addition to the list of instance Ids, the
        // request can also include the following properties:
        //     Force      When true, forces the instances to
        //                stop but you must check the integrity
        //                of the file system. Not recommended on
        //                Windows instances.
        //     Hibernate  When true, hibernates the instance if the
        //                instance was enabled for hibernation when
        //                it was launched.
        var request = new StopInstancesRequest
        {
            InstanceIds = new List<string> { ec2InstanceId },
        };

        var response = await _amazonEC2.StopInstancesAsync(request);

        if (response.StoppingInstances.Count > 0)
        {
            var instances = response.StoppingInstances;
            instances.ForEach(i =>
            {
                Console.WriteLine($"Successfully stopped the EC2 Instance " +
                                  $"with InstanceID: {i.InstanceId}.");
            });
        }
    }

    /// <summary>
    /// Terminate an EC2 instance.
    /// </summary>
    /// <param name="ec2InstanceId">The instance Id of the EC2 instance
    /// to terminate.</param>
    /// <returns>Async task.</returns>
    public async Task<List<InstanceStateChange>> TerminateInstances(string
ec2InstanceId)
    {
        var request = new TerminateInstancesRequest
        {
            InstanceIds = new List<string> { ec2InstanceId }
        };
```

```
        var response = await _amazonEC2.TerminateInstancesAsync(request);
        return response.TerminatingInstances;
    }

    /// <summary>
    /// Wait until an EC2 instance is in a specified state.
    /// </summary>
    /// <param name="instanceId">The instance Id.</param>
    /// <param name="stateName">The state to wait for.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> WaitForInstanceState(string instanceId,
 InstanceStateName stateName)
    {
        var request = new DescribeInstancesRequest
        {
            InstanceIds = new List<string> { instanceId }
        };

        // Wait until the instance is running.
        var hasState = false;
        do
        {
            // Wait 5 seconds.
            Thread.Sleep(5000);

            // Check for the desired state.
            var response = await _amazonEC2.DescribeInstancesAsync(request);
            var instance = response.Reservations[0].Instances[0];
            hasState = instance.State.Name == stateName;
            Console.Write(". ");
        } while (!hasState);

        return hasState;
    }

}
```

- 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。
  - AllocateAddress
  - AssociateAddress

- AuthorizeSecurityGroupIngress
- CreateKeyPair
- CreateSecurityGroup
- DeleteKeyPair
- DeleteSecurityGroup
- DescribeImages
- DescribeInstanceTypes
- DescribeInstances
- DescribeKeyPairs
- DescribeSecurityGroups
- DisassociateAddress
- ReleaseAddress
- RunInstances
- StartInstances
- StopInstances
- TerminateInstances
- UnmonitorInstances

を使用した Amazon ECS の例 AWS SDK for .NET

次のコード例は、Amazon ECS AWS SDK for .NET で 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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。このリンクには GitHub、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が記載されています。

開始方法

Hello Amazon ECS

次のコード例は、Amazon Cognito の使用を開始する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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.Hosting;

namespace ECSActions;

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 domain registration service.
        // Use your AWS profile name, or leave it blank to use the default profile.
        using var host = Host.CreateDefaultBuilder(args).Build();

        // Now the client is available for injection.
        var amazonECSClient = new AmazonECSClient();

        // You can use await and any of the async methods to get a response.
        var response = await amazonECSClient.ListClustersAsync(new
 ListClustersRequest { });

        Console.WriteLine($"Hello Amazon ECS! Following are some cluster ARNS
 available in the your aws account");
        Console.WriteLine();
        foreach (var arn in response.ClusterArns.Take(5))
        {
            Console.WriteLine($"\tARN: {arn}");
            Console.WriteLine($"Cluster Name: {arn.Split("/").Last()}");
            Console.WriteLine();
        }
```

```
    }
}
```

- API の詳細については、「 API リファレンスListClusters」の「」を参照してください。 AWS SDK for .NET

トピック

- アクション
- シナリオ

アクション

`ListClusters`

次の例は、`ListClusters` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// List cluster ARNs available.
    /// </summary>
    /// <returns>The ARN list of clusters.</returns>
    public async Task<List<string>> GetClusterARNSAsync()
    {

        Console.WriteLine("Getting a list of all the clusters in your AWS
account...");
        List<string> clusterArnList = new List<string>();
        // Get a list of all the clusters in your AWS account
        try
        {
```

```
            var listClustersResponse = _ecsClient.Paginators.ListClusters(new
ListClustersRequest
            {
            });

            var clusterArns = listClustersResponse.ClusterArns;

            // Print the ARNs of the clusters
            await foreach (var clusterArn in clusterArns)
            {
                clusterArnList.Add(clusterArn);
            }

            if (clusterArnList.Count == 0)
            {
                _logger.LogWarning("No clusters found in your AWS account.");
            }
            return clusterArnList;
        }
        catch (Exception e)
        {
            _logger.LogError($"An error occurred while getting a list of all the
clusters in your AWS account. {e.InnerException}");
            throw new Exception($"An error occurred while getting a list of all the
clusters in your AWS account. {e.InnerException}");
        }
    }
```

- API の詳細については、「 API リファレンスListClusters」の「」を参照してください。 AWS SDK for .NET

ListServices

次の例は、ListServices を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
/// List service ARNs available.
/// </summary>
/// <param name="clusterARN">The arn of the ECS cluster.</param>
/// <returns>The ARN list of services in given cluster.</returns>
public async Task<List<string>> GetServiceARNSAsync(string clusterARN)
{
    List<string> serviceArns = new List<string>();

    var request = new ListServicesRequest
    {
        Cluster = clusterARN
    };
    // Call the ListServices API operation and get the list of service ARNs
    var serviceList = _ecsClient.Paginators.ListServices(request);

    await foreach (var serviceARN in serviceList.ServiceArns)
    {
        if (serviceARN is null)
            continue;

        serviceArns.Add(serviceARN);
    }

    if (serviceArns.Count == 0)
    {
        _logger.LogWarning($"No services found in cluster {clusterARN} .");
    }

    return serviceArns;
}
```

- API の詳細については、「 API リファレンスListServices」の「」を参照してください。 AWS SDK for .NET

`ListTasks`

次の例は、`ListTasks` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
/// List task ARNs available.
/// </summary>
/// <param name="clusterARN">The arn of the ECS cluster.</param>
/// <returns>The ARN list of tasks in given cluster.</returns>
public async Task<List<string>> GetTaskARNsAsync(string clusterARN)
{
    // Set up the request to describe the tasks in the service
    var listTasksRequest = new ListTasksRequest
    {
        Cluster = clusterARN
    };
    List<string> taskArns = new List<string>();

    // Call the ListTasks API operation and get the list of task ARNs
    var tasks = _ecsClient.Paginators.ListTasks(listTasksRequest);

    await foreach (var task in tasks.TaskArns)
    {
        if (task is null)
            continue;


        taskArns.Add(task);
    }

    if (taskArns.Count == 0)
```

```
        {
            _logger.LogWarning("No tasks found in cluster: " + clusterARN);
        }

        return taskArns;
    }
```

- API の詳細については、「 API リファレンスListTasks」の「」を参照してください。 AWS SDK for .NET

シナリオ

クラスター、サービス、タスクの ARN 情報を取得する

次のコードサンプルは、以下の操作方法を示しています。

- すべてのクラスターのリストを取得する。
- クラスターのサービスを取得する。
- クラスターのタスクを取得する。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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のシナリオを実行します。

```
using Amazon.ECS;
using ECSActions;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;

namespace ECSScenario;
```

```
public class ECSScenario
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.


    This .NET example performs the following tasks:
        1. List ECS Cluster ARNs.
        2. List services in every cluster
        3. List Task ARNs in every cluster.
    */

    private static ILogger logger = null!;
    private static ECSWrapper _ecsWrapper = 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))
        .Build();

        ILoggerFactory loggerFactory = LoggerFactory.Create(builder =>
        {
            builder.AddConsole();
        });

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<ECSScenario>();

        var loggerECSWarpper = LoggerFactory.Create(builder =>
 { builder.AddConsole(); })
          .CreateLogger<ECSWrapper>();

        var amazonECSClient = new AmazonECSClient();

        _ecsWrapper = new ECSWrapper(amazonECSClient, loggerECSWarpper);

        Console.WriteLine(new string('-', 80));
```

```
        Console.WriteLine("Welcome to the Amazon ECS example scenario.");
        Console.WriteLine(new string('-', 80));

        try
        {
            await ListClusterARNs();
            await ListServiceARNs();
            await ListTaskARNs();

        }
        catch (Exception ex)
        {
            logger.LogError(ex, "There was a problem executing the scenario.");
        }
    }

    /// <summary>
    /// List ECS Cluster ARNs
    /// </summary>
    private static async Task ListClusterARNs()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"1. List Cluster ARNs from ECS.");
        var arns = await _ecsWrapper.GetClusterARNSAsync();

        foreach (var arn in arns)
        {
            Console.WriteLine($"Cluster arn: {arn}");
            Console.WriteLine($"Cluster name: {arn.Split("/").Last()}");
        }

        Console.WriteLine(new string('-', 80));
    }


    /// <summary>
    /// List services in every cluster
    /// </summary>
    private static async Task ListServiceARNs()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"2. List Service ARNs in every cluster.");
        var clusterARNs = await _ecsWrapper.GetClusterARNSAsync();
```

```
        foreach (var clusterARN in clusterARNs)
        {
            Console.WriteLine($"Getting services for cluster name:
{clusterARN.Split("/").Last()}");
            Console.WriteLine(new string('.', 5));


            var serviceARNs = await _ecsWrapper.GetServiceARNSAsync(clusterARN);

            foreach (var serviceARN in serviceARNs)
            {
                Console.WriteLine($"Service arn: {serviceARN}");
                Console.WriteLine($"Service name: {serviceARN.Split("/").Last()}");
            }
        }

        Console.WriteLine(new string('-', 80));
    }


    /// <summary>
    /// List tasks in every cluster
    /// </summary>
    private static async Task ListTaskARNs()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"3. List Task ARNs in every cluster.");
        var clusterARNs = await _ecsWrapper.GetClusterARNSAsync();

        foreach (var clusterARN in clusterARNs)
        {
            Console.WriteLine($"Getting tasks for cluster name:
{clusterARN.Split("/").Last()}");
            Console.WriteLine(new string('.', 5));

            var taskARNs = await _ecsWrapper.GetTaskARNsAsync(clusterARN);

            foreach (var taskARN in taskARNs)
            {
                Console.WriteLine($"Task arn: {taskARN}");
            }
        }
        Console.WriteLine(new string('-', 80));
    }
```

```
}
```

Amazon ECS 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.ECS;
using Amazon.ECS.Model;
using Microsoft.Extensions.Logging;

namespace ECSActions;

public class ECSWrapper
{
    private readonly AmazonECSClient _ecsClient;
    private readonly ILogger<ECSWrapper> _logger;

    /// <summary>
    /// Constructor for the ECS wrapper.
    /// </summary>
    /// <param name="ecsClient">The injected ECS client.</param>
    /// <param name="logger">The injected logger for the wrapper.</param>
    public ECSWrapper(AmazonECSClient ecsClient, ILogger<ECSWrapper> logger)

    {
        _logger = logger;
        _ecsClient = ecsClient;
    }

    /// <summary>
    /// List cluster ARNs available.
    /// </summary>
    /// <returns>The ARN list of clusters.</returns>
    public async Task<List<string>> GetClusterARNSAsync()
    {

        Console.WriteLine("Getting a list of all the clusters in your AWS
 account...");
        List<string> clusterArnList = new List<string>();
        // Get a list of all the clusters in your AWS account
        try
        {
```

```
            var listClustersResponse = _ecsClient.Paginators.ListClusters(new
ListClustersRequest
            {
            });

            var clusterArns = listClustersResponse.ClusterArns;

            // Print the ARNs of the clusters
            await foreach (var clusterArn in clusterArns)
            {
                clusterArnList.Add(clusterArn);
            }

            if (clusterArnList.Count == 0)
            {
                _logger.LogWarning("No clusters found in your AWS account.");
            }
            return clusterArnList;
        }
        catch (Exception e)
        {
            _logger.LogError($"An error occurred while getting a list of all the
clusters in your AWS account. {e.InnerException}");
            throw new Exception($"An error occurred while getting a list of all the
clusters in your AWS account. {e.InnerException}");
        }
    }

    /// <summary>
    /// List service ARNs available.
    /// </summary>
    /// <param name="clusterARN">The arn of the ECS cluster.</param>
    /// <returns>The ARN list of services in given cluster.</returns>
    public async Task<List<string>> GetServiceARNSAsync(string clusterARN)
    {
        List<string> serviceArns = new List<string>();

        var request = new ListServicesRequest
        {
            Cluster = clusterARN
        };
        // Call the ListServices API operation and get the list of service ARNs
        var serviceList = _ecsClient.Paginators.ListServices(request);
```

```
        await foreach (var serviceARN in serviceList.ServiceArns)
        {
            if (serviceARN is null)
                continue;

            serviceArns.Add(serviceARN);
        }

        if (serviceArns.Count == 0)
        {
            _logger.LogWarning($"No services found in cluster {clusterARN} .");
        }

        return serviceArns;
    }

    /// <summary>
    /// List task ARNs available.
    /// </summary>
    /// <param name="clusterARN">The arn of the ECS cluster.</param>
    /// <returns>The ARN list of tasks in given cluster.</returns>
    public async Task<List<string>> GetTaskARNsAsync(string clusterARN)
    {
        // Set up the request to describe the tasks in the service
        var listTasksRequest = new ListTasksRequest
        {
            Cluster = clusterARN
        };
        List<string> taskArns = new List<string>();

        // Call the ListTasks API operation and get the list of task ARNs
        var tasks = _ecsClient.Paginators.ListTasks(listTasksRequest);

        await foreach (var task in tasks.TaskArns)
        {
            if (task is null)
                continue;


            taskArns.Add(task);
        }

        if (taskArns.Count == 0)
        {
```

```
            _logger.LogWarning("No tasks found in cluster: " + clusterARN);
        }

        return taskArns;
    }
}
```

- 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。
  - ListClusters
  - ListServices
  - ListTasks

を使用した Elastic Load Balancing - バージョン 2 の例 AWS SDK for .NET

次のコード例は、Elastic Load Balancing - バージョン 2 AWS SDK for .NET で 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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。このリンクには GitHub、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が記載されています。

トピック

- アクション
- シナリオ

アクション

`CreateListener`

次の例は、`CreateListener` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
   /// <summary>
   /// Create an Elastic Load Balancing load balancer that uses the specified
subnets
   /// and forwards requests to the specified target group.
   /// </summary>
   /// <param name="name">The name for the new load balancer.</param>
   /// <param name="subnetIds">Subnets for the load balancer.</param>
   /// <param name="targetGroup">Target group for forwarded requests.</param>
   /// <returns>The new LoadBalancer object.</returns>
   public async Task<LoadBalancer> CreateLoadBalancerAndListener(string name,
List<string> subnetIds, TargetGroup targetGroup)
   {
       var createLbResponse = await
_amazonElasticLoadBalancingV2.CreateLoadBalancerAsync(
           new CreateLoadBalancerRequest()
           {
               Name = name,
               Subnets = subnetIds
           });
       var loadBalancerArn = createLbResponse.LoadBalancers[0].LoadBalancerArn;

       // Wait for load balancer to be available.
       var loadBalancerReady = false;
       while (!loadBalancerReady)
       {
           try
           {
               var describeResponse =
                   await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                       new DescribeLoadBalancersRequest()
                       {
                           Names = new List<string>() { name }
                       });
```

```
                var loadBalancerState =
describeResponse.LoadBalancers[0].State.Code;

                loadBalancerReady = loadBalancerState ==
LoadBalancerStateEnum.Active;
            }
            catch (LoadBalancerNotFoundException)
            {
                loadBalancerReady = false;
            }
            Thread.Sleep(10000);
        }
        // Create the listener.
        await _amazonElasticLoadBalancingV2.CreateListenerAsync(
            new CreateListenerRequest()
            {
                LoadBalancerArn = loadBalancerArn,
                Protocol = targetGroup.Protocol,
                Port = targetGroup.Port,
                DefaultActions = new List<Action>()
                {
                    new Action()
                    {
                        Type = ActionTypeEnum.Forward,
                        TargetGroupArn = targetGroup.TargetGroupArn
                    }
                }
            });
        return createLbResponse.LoadBalancers[0];
    }
```

- API の詳細については、「 API リファレンスCreateListener」の「」を参照してください。AWS SDK for .NET

`CreateLoadBalancer`

次の例は、`CreateLoadBalancer` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
   /// <summary>
   /// Create an Elastic Load Balancing load balancer that uses the specified
subnets
   /// and forwards requests to the specified target group.
   /// </summary>
   /// <param name="name">The name for the new load balancer.</param>
   /// <param name="subnetIds">Subnets for the load balancer.</param>
   /// <param name="targetGroup">Target group for forwarded requests.</param>
   /// <returns>The new LoadBalancer object.</returns>
   public async Task<LoadBalancer> CreateLoadBalancerAndListener(string name,
List<string> subnetIds, TargetGroup targetGroup)
   {
       var createLbResponse = await
_amazonElasticLoadBalancingV2.CreateLoadBalancerAsync(
           new CreateLoadBalancerRequest()
           {
               Name = name,
               Subnets = subnetIds
           });
       var loadBalancerArn = createLbResponse.LoadBalancers[0].LoadBalancerArn;

       // Wait for load balancer to be available.
       var loadBalancerReady = false;
       while (!loadBalancerReady)
       {
           try
           {
               var describeResponse =
                   await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                       new DescribeLoadBalancersRequest()
                       {
                           Names = new List<string>() { name }
                       });
```

```
                var loadBalancerState =
describeResponse.LoadBalancers[0].State.Code;

                loadBalancerReady = loadBalancerState ==
LoadBalancerStateEnum.Active;
            }
            catch (LoadBalancerNotFoundException)
            {
                loadBalancerReady = false;
            }
            Thread.Sleep(10000);
        }
        // Create the listener.
        await _amazonElasticLoadBalancingV2.CreateListenerAsync(
            new CreateListenerRequest()
            {
                LoadBalancerArn = loadBalancerArn,
                Protocol = targetGroup.Protocol,
                Port = targetGroup.Port,
                DefaultActions = new List<Action>()
                {
                    new Action()
                    {
                        Type = ActionTypeEnum.Forward,
                        TargetGroupArn = targetGroup.TargetGroupArn
                    }
                }
            });
        return createLbResponse.LoadBalancers[0];
    }
```

- API の詳細については、「 API リファレンスCreateLoadBalancer」の「」を参照してください。 AWS SDK for .NET

`CreateTargetGroup`

次の例は、`CreateTargetGroup` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Create an Elastic Load Balancing target group. The target group specifies
 how the load balancer forwards
    /// requests to instances in the group and how instance health is checked.
    ///
    /// To speed up this demo, the health check is configured with shortened times
 and lower thresholds. In production,
    /// you might want to decrease the sensitivity of your health checks to avoid
 unwanted failures.
    /// </summary>
    /// <param name="groupName">The name for the group.</param>
    /// <param name="protocol">The protocol, such as HTTP.</param>
    /// <param name="port">The port to use to forward requests, such as 80.</param>
    /// <param name="vpcId">The Id of the Vpc in which the load balancer exists.</
param>
    /// <returns>The new TargetGroup object.</returns>
    public async Task<TargetGroup> CreateTargetGroupOnVpc(string groupName,
 ProtocolEnum protocol, int port, string vpcId)
    {
        var createResponse = await
 _amazonElasticLoadBalancingV2.CreateTargetGroupAsync(
            new CreateTargetGroupRequest()
            {
                Name = groupName,
                Protocol = protocol,
                Port = port,
                HealthCheckPath = "/healthcheck",
                HealthCheckIntervalSeconds = 10,
                HealthCheckTimeoutSeconds = 5,
                HealthyThresholdCount = 2,
                UnhealthyThresholdCount = 2,
                VpcId = vpcId
            });
        var targetGroup = createResponse.TargetGroups[0];
```

```
        return targetGroup;
    }
```

- API の詳細については、「 API リファレンスCreateTargetGroup」の「」を参照してください。 AWS SDK for .NET

`DeleteLoadBalancer`

次の例は、`DeleteLoadBalancer` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete a load balancer by its specified name.
    /// </summary>
    /// <param name="name">The name of the load balancer to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteLoadBalancerByName(string name)
    {
        try
        {
            var describeLoadBalancerResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                    new DescribeLoadBalancersRequest()
                    {
                        Names = new List<string>() { name }
                    });
            var lbArn =
describeLoadBalancerResponse.LoadBalancers[0].LoadBalancerArn;
            await _amazonElasticLoadBalancingV2.DeleteLoadBalancerAsync(
                new DeleteLoadBalancerRequest()
                {
                    LoadBalancerArn = lbArn
                }
```

```
            );
        }
        catch (LoadBalancerNotFoundException)
        {
            Console.WriteLine($"Load balancer {name} not found.");
        }
    }
```

- API の詳細については、「 API リファレンスDeleteLoadBalancer」の「」を参照してください。 AWS SDK for .NET

`DeleteTargetGroup`

次の例は、DeleteTargetGroup を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
/// Delete a TargetGroup by its specified name.
/// </summary>
/// <param name="groupName">Name of the group to delete.</param>
/// <returns>Async task.</returns>
public async Task DeleteTargetGroupByName(string groupName)
{
    var done = false;
    while (!done)
    {
        try
        {
            var groupResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                    new DescribeTargetGroupsRequest()
                    {
                        Names = new List<string>() { groupName }
```

```
                    });

                var targetArn = groupResponse.TargetGroups[0].TargetGroupArn;
                await _amazonElasticLoadBalancingV2.DeleteTargetGroupAsync(
                    new DeleteTargetGroupRequest() { TargetGroupArn = targetArn });
                Console.WriteLine($"Deleted load balancing target group
{groupName}.");
                done = true;
            }
            catch (TargetGroupNotFoundException)
            {
                Console.WriteLine(
                    $"Target group {groupName} not found, could not delete.");
                done = true;
            }
            catch (ResourceInUseException)
            {
                Console.WriteLine("Target group not yet released, waiting...");
                Thread.Sleep(10000);
            }
        }
    }
```

- API の詳細については、「 API リファレンスDeleteTargetGroup」の「」を参照してください。 AWS SDK for .NET

`DescribeLoadBalancers`

次の例は、`DescribeLoadBalancers` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get the HTTP Endpoint of a load balancer by its name.
```

```
    /// </summary>
    /// <param name="loadBalancerName">The name of the load balancer.</param>
    /// <returns>The HTTP endpoint.</returns>
    public async Task<string> GetEndpointForLoadBalancerByName(string
loadBalancerName)
    {
        if (_endpoint == null)
        {
            var endpointResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                    new DescribeLoadBalancersRequest()
                    {
                        Names = new List<string>() { loadBalancerName }
                    });
            _endpoint = endpointResponse.LoadBalancers[0].DNSName;
        }

        return _endpoint;
    }
```

- API の詳細については、「 API リファレンスDescribeLoadBalancers」の「」を参照してください。 AWS SDK for .NET

`DescribeTargetHealth`

次の例は、`DescribeTargetHealth` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get the target health for a group by name.
    /// </summary>
    /// <param name="groupName">The name of the group.</param>
    /// <returns>The collection of health descriptions.</returns>
```

```
    public async Task<List<TargetHealthDescription>>
CheckTargetHealthForGroup(string groupName)
    {
        List<TargetHealthDescription> result = null!;
        try
        {
            var groupResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                    new DescribeTargetGroupsRequest()
                    {
                        Names = new List<string>() { groupName }
                    });
            var healthResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetHealthAsync(
                    new DescribeTargetHealthRequest()
                    {
                        TargetGroupArn =
groupResponse.TargetGroups[0].TargetGroupArn
                    });
            ;
            result = healthResponse.TargetHealthDescriptions;
        }
        catch (TargetGroupNotFoundException)
        {
            Console.WriteLine($"Target group {groupName} not found.");
        }
        return result;
    }
```

- API の詳細については、「 API リファレンスDescribeTargetHealth」の「」を参照してください。 AWS SDK for .NET

シナリオ

レジリエントなサービスの構築と管理

次のコード例は、本、映画、曲のレコメンデーションを返す負荷分散型ウェブサービスの作成方法を示しています。この例は、障害に対するサービスの対応方法と、障害発生時の耐障害性を高めるためにサービスを再構築する方法を示しています。

- Amazon EC2 Auto Scaling グループを使用して、起動テンプレートに基づいて Amazon Elastic Compute Cloud (Amazon EC2) インスタンスを作成し、インスタンス数を所定の範囲内に維持します。
- Elastic Load Balancing で HTTP リクエストを処理して配信します。
- Auto Scaling グループ内のインスタンスの状態を監視し、正常なインスタンスにのみリクエストを転送します。
- 各 EC2 インスタンスで Python ウェブサーバーを実行して HTTP リクエストを処理します。ウェブサーバーはレコメンデーションとヘルスチェックを返します。
- Amazon DynamoDB テーブルを使用してレコメンデーションサービスをシミュレートできます。
- AWS Systems Manager パラメータを更新して、リクエストとヘルスチェックに対するウェブサーバーの応答を制御します。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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のシナリオを実行します。

```
    static async Task Main(string[] args)
    {
        _configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally, load local settings.
            .Build();


        // Set up dependency injection for the AWS services.
        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<IAmazonIdentityManagementService>()
                    .AddAWSService<IAmazonDynamoDB>()
                    .AddAWSService<IAmazonElasticLoadBalancingV2>()
                    .AddAWSService<IAmazonSimpleSystemsManagement>()
                    .AddAWSService<IAmazonAutoScaling>()
                    .AddAWSService<IAmazonEC2>()
                    .AddTransient<AutoScalerWrapper>()
                    .AddTransient<ElasticLoadBalancerWrapper>()
                    .AddTransient<SmParameterWrapper>()
                    .AddTransient<Recommendations>()
                    .AddSingleton<IConfiguration>(_configuration)
            )
            .Build();

        ServicesSetup(host);
        ResourcesSetup();

        try
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Welcome to the Resilient Architecture Example
Scenario.");
            Console.WriteLine(new string('-', 80));
            await Deploy(true);

            Console.WriteLine("Now let's begin the scenario.");
            Console.WriteLine(new string('-', 80));
            await Demo(true);

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Finally, let's clean up our resources.");
            Console.WriteLine(new string('-', 80));

            await DestroyResources(true);

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Resilient Architecture Example Scenario is
complete.");
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            Console.WriteLine(new string('-', 80));
```

```
            Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
            await DestroyResources(true);
            Console.WriteLine(new string('-', 80));
        }
    }

    /// <summary>
    /// Setup any common resources, also used for integration testing.
    /// </summary>
    public static void ResourcesSetup()
    {
        _httpClient = new HttpClient();
    }

    /// <summary>
    /// Populate the services for use within the console application.
    /// </summary>
    /// <param name="host">The services host.</param>
    private static void ServicesSetup(IHost host)
    {
        _elasticLoadBalancerWrapper =
host.Services.GetRequiredService<ElasticLoadBalancerWrapper>();
        _iamClient =
host.Services.GetRequiredService<IAmazonIdentityManagementService>();
        _recommendations = host.Services.GetRequiredService<Recommendations>();
        _autoScalerWrapper = host.Services.GetRequiredService<AutoScalerWrapper>();
        _smParameterWrapper =
host.Services.GetRequiredService<SmParameterWrapper>();
    }

    /// <summary>
    /// Deploy necessary resources for the scenario.
    /// </summary>
    /// <param name="interactive">True to run as interactive.</param>
    /// <returns>True if successful.</returns>
    public static async Task<bool> Deploy(bool interactive)
    {
        var protocol = "HTTP";
        var port = 80;
        var sshPort = 22;

        Console.WriteLine(
```

```
            "\nFor this demo, we'll use the AWS SDK for .NET to create several AWS
resources\n" +
            "to set up a load-balanced web service endpoint and explore some ways to
make it resilient\n" +
            "against various kinds of failures.\n\n" +
            "Some of the resources create by this demo are:\n");

        Console.WriteLine(
            "\t* A DynamoDB table that the web service depends on to provide book,
movie, and song recommendations.");
        Console.WriteLine(
            "\t* An EC2 launch template that defines EC2 instances that each contain
a Python web server.");
        Console.WriteLine(
            "\t* An EC2 Auto Scaling group that manages EC2 instances across several
Availability Zones.");
        Console.WriteLine(
            "\t* An Elastic Load Balancing (ELB) load balancer that targets the Auto
Scaling group to distribute requests.");
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to start deploying
resources.");
        if (interactive)
            Console.ReadLine();

        // Create and populate the DynamoDB table.
        var databaseTableName = _configuration["databaseName"];
        var recommendationsPath = Path.Join(_configuration["resourcePath"],
            "recommendations_objects.json");
        Console.WriteLine($"Creating and populating a DynamoDB table named
{databaseTableName}.");
        await _recommendations.CreateDatabaseWithName(databaseTableName);
        await _recommendations.PopulateDatabase(databaseTableName,
recommendationsPath);
        Console.WriteLine(new string('-', 80));

        // Create the EC2 Launch Template.

        Console.WriteLine(
            $"Creating an EC2 launch template that runs 'server_startup_script.sh'
when an instance starts.\n"
            + "\nThis script starts a Python web server defined in the `server.py`
script. The web server\n"
```

```
            + "listens to HTTP requests on port 80 and responds to requests to '/'
and to '/healthcheck'.\n"
            + "For demo purposes, this server is run as the root user. In
production, the best practice is to\n"
            + "run a web server, such as Apache, with least-privileged
credentials.");
        Console.WriteLine(
            "\nThe template also defines an IAM policy that each instance uses to
assume a role that grants\n"
            + "permissions to access the DynamoDB recommendation table and Systems
Manager parameters\n"
            + "that control the flow of the demo.");

        var startupScriptPath = Path.Join(_configuration["resourcePath"],
            "server_startup_script.sh");
        var instancePolicyPath = Path.Join(_configuration["resourcePath"],
            "instance_policy.json");
        await _autoScalerWrapper.CreateTemplate(startupScriptPath,
instancePolicyPath);
        Console.WriteLine(new string('-', 80));

        Console.WriteLine(
            "Creating an EC2 Auto Scaling group that maintains three EC2 instances,
each in a different\n"
            + "Availability Zone.\n");
        var zones = await _autoScalerWrapper.DescribeAvailabilityZones();
        await _autoScalerWrapper.CreateGroupOfSize(3, _autoScalerWrapper.GroupName,
zones);
        Console.WriteLine(new string('-', 80));

        Console.WriteLine(
            "At this point, you have EC2 instances created. Once each instance
starts, it listens for\n"
            + "HTTP requests. You can see these instances in the console or continue
with the demo.\n");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to continue.");
        if (interactive)
            Console.ReadLine();

        Console.WriteLine("Creating variables that control the flow of the demo.");
        await _smParameterWrapper.Reset();
```

```
        Console.WriteLine(
            "\nCreating an Elastic Load Balancing target group and load balancer.
 The target group\n"
            + "defines how the load balancer connects to instances. The load
 balancer provides a\n"
            + "single endpoint where clients connect and dispatches requests to
 instances in the group.");

        var defaultVpc = await _autoScalerWrapper.GetDefaultVpc();
        var subnets = await
 _autoScalerWrapper.GetAllVpcSubnetsForZones(defaultVpc.VpcId, zones);
        var subnetIds = subnets.Select(s => s.SubnetId).ToList();
        var targetGroup = await
 _elasticLoadBalancerWrapper.CreateTargetGroupOnVpc(_elasticLoadBalancerWrapper.TargetGroupN
 protocol, port, defaultVpc.VpcId);

        await
 _elasticLoadBalancerWrapper.CreateLoadBalancerAndListener(_elasticLoadBalancerWrapper.LoadB
 subnetIds, targetGroup);
        await
 _autoScalerWrapper.AttachLoadBalancerToGroup(_autoScalerWrapper.GroupName,
 targetGroup.TargetGroupArn);
        Console.WriteLine("\nVerifying access to the load balancer endpoint...");
        var endPoint = await
 _elasticLoadBalancerWrapper.GetEndpointForLoadBalancerByName(_elasticLoadBalancerWrapper.Lo
        var loadBalancerAccess = await
 _elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);

        if (!loadBalancerAccess)
        {
            Console.WriteLine("\nCouldn't connect to the load balancer, verifying
 that the port is open...");

            var ipString = await _httpClient.GetStringAsync("https://
checkip.amazonaws.com");
            ipString = ipString.Trim();

            var defaultSecurityGroup = await
 _autoScalerWrapper.GetDefaultSecurityGroupForVpc(defaultVpc);
            var portIsOpen =
 _autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, port, ipString);
            var sshPortIsOpen =
 _autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, sshPort,
 ipString);
```

```
            if (!portIsOpen)
            {
                Console.WriteLine(
                    "\nFor this example to work, the default security group for your
 default VPC must\n"
                    + "allows access from this computer. You can either add it
 automatically from this\n"
                    + "example or add it yourself using the AWS Management Console.
\n");

                if (!interactive || GetYesNoResponse(
                        "Do you want to add a rule to the security group to allow
 inbound traffic from your computer's IP address?"))
                {
                    await
 _autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, port, ipString);
                }
            }

            if (!sshPortIsOpen)
            {
                if (!interactive || GetYesNoResponse(
                        "Do you want to add a rule to the security group to allow
 inbound SSH traffic for debugging from your computer's IP address?"))
                {
                    await
 _autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, sshPort,
 ipString);
                }
            }
            loadBalancerAccess = await
 _elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);
        }

        if (loadBalancerAccess)
        {
            Console.WriteLine("Your load balancer is ready. You can access it by
 browsing to:");
            Console.WriteLine($"\thttp://{endPoint}\n");
        }
        else
        {
            Console.WriteLine(
```

```
                "\nCouldn't get a successful response from the load balancer
endpoint. Troubleshoot by\n"
                + "manually verifying that your VPC and security group are
configured correctly and that\n"
                + "you can successfully make a GET request to the load balancer
endpoint:\n");
            Console.WriteLine($"\thttp://{endPoint}\n");
        }
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to continue with the
demo.");
        if (interactive)
            Console.ReadLine();
        return true;
    }

    /// <summary>
    /// Demonstrate the steps of the scenario.
    /// </summary>
    /// <param name="interactive">True to run as an interactive scenario.</param>
    /// <returns>Async task.</returns>
    public static async Task<bool> Demo(bool interactive)
    {
        var ssmOnlyPolicy = Path.Join(_configuration["resourcePath"],
            "ssm_only_policy.json");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Resetting parameters to starting values for demo.");
        await _smParameterWrapper.Reset();

        Console.WriteLine("\nThis part of the demonstration shows how to toggle
different parts of the system\n" +
                          "to create situations where the web service fails, and
shows how using a resilient\n" +
                          "architecture can keep the web service running in spite of
these failures.");
        Console.WriteLine(new string('-', 88));
        Console.WriteLine("At the start, the load balancer endpoint returns
recommendations and reports that all targets are healthy.");
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine($"The web service running on the EC2 instances gets
recommendations by querying a DynamoDB table.\n" +
```

```
                        $"The table name is contained in a Systems Manager
 parameter named '{_smParameterWrapper.TableParameter}'.\n" +
                        $"To simulate a failure of the recommendation service,
 let's set this parameter to name a non-existent table.\n");
        await
 _smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter, "this-
is-not-a-table");
        Console.WriteLine("\nNow, sending a GET request to the load balancer
 endpoint returns a failure code. But, the service reports as\n" +
                        "healthy to the load balancer because shallow health
 checks don't check for failure of the recommendation service.");
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("Instead of failing when the recommendation service fails,
 the web service can return a static response.");
        Console.WriteLine("While this is not a perfect solution, it presents the
 customer with a somewhat better experience than failure.");

        await
 _smParameterWrapper.PutParameterByName(_smParameterWrapper.FailureResponseParameter,
 "static");

        Console.WriteLine("\nNow, sending a GET request to the load balancer
 endpoint returns a static response.");
        Console.WriteLine("The service still reports as healthy because health
 checks are still shallow.");
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("Let's reinstate the recommendation service.\n");
        await
 _smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter,
 _smParameterWrapper.TableName);
        Console.WriteLine(
            "\nLet's also substitute bad credentials for one of the instances in the
 target group so that it can't\n" +
            "access the DynamoDB recommendation table.\n"
        );
        await _autoScalerWrapper.CreateInstanceProfileWithName(
            _autoScalerWrapper.BadCredsPolicyName,
            _autoScalerWrapper.BadCredsRoleName,
            _autoScalerWrapper.BadCredsProfileName,
            ssmOnlyPolicy,
```

```
            new List<string> { "AmazonSSMManagedInstanceCore" }
        );
        var instances = await
_autoScalerWrapper.GetInstancesByGroupName(_autoScalerWrapper.GroupName);
        var badInstanceId = instances.First();
        var instanceProfile = await
_autoScalerWrapper.GetInstanceProfile(badInstanceId);
        Console.WriteLine(
            $"Replacing the profile for instance {badInstanceId} with a profile that
contains\n" +
            "bad credentials...\n"
        );
        await _autoScalerWrapper.ReplaceInstanceProfile(
            badInstanceId,
            _autoScalerWrapper.BadCredsProfileName,
            instanceProfile.AssociationId
        );
        Console.WriteLine(
            "Now, sending a GET request to the load balancer endpoint returns either
a recommendation or a static response,\n" +
            "depending on which instance is selected by the load balancer.\n"
        );
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nLet's implement a deep health check. For this demo, a
deep health check tests whether");
        Console.WriteLine("the web service can access the DynamoDB table that it
depends on for recommendations. Note that");
        Console.WriteLine("the deep health check is only for ELB routing and not for
Auto Scaling instance health.");
        Console.WriteLine("This kind of deep health check is not recommended for
Auto Scaling instance health, because it");
        Console.WriteLine("risks accidental termination of all instances in the Auto
Scaling group when a dependent service fails.");

        Console.WriteLine("\nBy implementing deep health checks, the load balancer
can detect when one of the instances is failing");
        Console.WriteLine("and take that instance out of rotation.");

        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.HealthCheckParameter,
"deep");
```

```
        Console.WriteLine($"\nNow, checking target health indicates that the
 instance with bad credentials ({badInstanceId})");
        Console.WriteLine("is unhealthy. Note that it might take a minute or two for
 the load balancer to detect the unhealthy");
        Console.WriteLine("instance. Sending a GET request to the load balancer
 endpoint always returns a recommendation, because");
        Console.WriteLine("the load balancer takes unhealthy instances out of its
 rotation.");

        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nBecause the instances in this demo are controlled by an
 auto scaler, the simplest way to fix an unhealthy");
        Console.WriteLine("instance is to terminate it and let the auto scaler start
 a new instance to replace it.");

        await _autoScalerWrapper.TryTerminateInstanceById(badInstanceId);

        Console.WriteLine($"\nEven while the instance is terminating and the new
 instance is starting, sending a GET");
        Console.WriteLine("request to the web service continues to get a successful
 recommendation response because");
        Console.WriteLine("starts and reports as healthy, it is included in the load
 balancing rotation.");
        Console.WriteLine("Note that terminating and replacing an instance typically
 takes several minutes, during which time you");
        Console.WriteLine("can see the changing health check status until the new
 instance is running and healthy.");

        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nIf the recommendation service fails now, deep health
 checks mean all instances report as unhealthy.");

        await
 _smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter, "this-
is-not-a-table");

        Console.WriteLine($"\nWhen all instances are unhealthy, the load balancer
 continues to route requests even to");
        Console.WriteLine("unhealthy instances, allowing them to fail open and
 return a static response rather than fail");
```

```
        Console.WriteLine("closed and report failure to the customer.");

        if (interactive)
            await DemoActionChoices();
        await _smParameterWrapper.Reset();

        Console.WriteLine(new string('-', 80));
        return true;
    }

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <param name="interactive">True to ask the user for cleanup.</param>
    /// <returns>Async task.</returns>
    public static async Task<bool> DestroyResources(bool interactive)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine(
            "To keep things tidy and to avoid unwanted charges on your account, we
can clean up all AWS resources\n" +
            "that were created for this demo."
        );

        if (!interactive || GetYesNoResponse("Do you want to clean up all demo
resources? (y/n) "))
        {
            await
_elasticLoadBalancerWrapper.DeleteLoadBalancerByName(_elasticLoadBalancerWrapper.LoadBalanc
            await
_elasticLoadBalancerWrapper.DeleteTargetGroupByName(_elasticLoadBalancerWrapper.TargetGroup
            await
_autoScalerWrapper.TerminateAndDeleteAutoScalingGroupWithName(_autoScalerWrapper.GroupName)
            await
_autoScalerWrapper.DeleteKeyPairByName(_autoScalerWrapper.KeyPairName);
            await
_autoScalerWrapper.DeleteTemplateByName(_autoScalerWrapper.LaunchTemplateName);
            await _autoScalerWrapper.DeleteInstanceProfile(
                _autoScalerWrapper.BadCredsProfileName,
                _autoScalerWrapper.BadCredsRoleName
            );
            await
_recommendations.DestroyDatabaseByName(_recommendations.TableName);
        }
```

```
        else
        {
            Console.WriteLine(
                "Ok, we'll leave the resources intact.\n" +
                "Don't forget to delete them when you're done with them or you might
 incur unexpected charges."
            );
        }

        Console.WriteLine(new string('-', 80));
        return true;
    }
```

Auto Scaling と Amazon EC2 のアクションをラップするクラスを作成します。

```
/// <summary>
/// Encapsulates Amazon EC2 Auto Scaling and EC2 management methods.
/// </summary>
public class AutoScalerWrapper
{
    private readonly IAmazonAutoScaling _amazonAutoScaling;
    private readonly IAmazonEC2 _amazonEc2;
    private readonly IAmazonSimpleSystemsManagement _amazonSsm;
    private readonly IAmazonIdentityManagementService _amazonIam;

    private readonly string _instanceType = "";
    private readonly string _amiParam = "";
    private readonly string _launchTemplateName = "";
    private readonly string _groupName = "";
    private readonly string _instancePolicyName = "";
    private readonly string _instanceRoleName = "";
    private readonly string _instanceProfileName = "";
    private readonly string _badCredsProfileName = "";
    private readonly string _badCredsRoleName = "";
    private readonly string _badCredsPolicyName = "";
    private readonly string _keyPairName = "";

    public string GroupName => _groupName;
    public string KeyPairName => _keyPairName;
    public string LaunchTemplateName => _launchTemplateName;
    public string InstancePolicyName => _instancePolicyName;
    public string BadCredsProfileName => _badCredsProfileName;
```

```
    public string BadCredsRoleName => _badCredsRoleName;
    public string BadCredsPolicyName => _badCredsPolicyName;

    /// <summary>
    /// Constructor for the AutoScalerWrapper.
    /// </summary>
    /// <param name="amazonAutoScaling">The injected AutoScaling client.</param>
    /// <param name="amazonEc2">The injected EC2 client.</param>
    /// <param name="amazonIam">The injected IAM client.</param>
    /// <param name="amazonSsm">The injected SSM client.</param>
    public AutoScalerWrapper(
        IAmazonAutoScaling amazonAutoScaling,
        IAmazonEC2 amazonEc2,
        IAmazonSimpleSystemsManagement amazonSsm,
        IAmazonIdentityManagementService amazonIam,
        IConfiguration configuration)
    {
        _amazonAutoScaling = amazonAutoScaling;
        _amazonEc2 = amazonEc2;
        _amazonSsm = amazonSsm;
        _amazonIam = amazonIam;

        var prefix = configuration["resourcePrefix"];
        _instanceType = configuration["instanceType"];
        _amiParam = configuration["amiParam"];

        _launchTemplateName = prefix + "-template";
        _groupName = prefix + "-group";
        _instancePolicyName = prefix + "-pol";
        _instanceRoleName = prefix + "-role";
        _instanceProfileName = prefix + "-prof";
        _badCredsPolicyName = prefix + "-bc-pol";
        _badCredsRoleName = prefix + "-bc-role";
        _badCredsProfileName = prefix + "-bc-prof";
        _keyPairName = prefix + "-key-pair";
    }

    /// <summary>
    /// Create a policy, role, and profile that is associated with instances with a
specified name.
    /// An instance's associated profile defines a role that is assumed by the
    /// instance.The role has attached policies that specify the AWS permissions
granted to
    /// clients that run on the instance.
```

```
    /// </summary>
    /// <param name="policyName">Name to use for the policy.</param>
    /// <param name="roleName">Name to use for the role.</param>
    /// <param name="profileName">Name to use for the profile.</param>
    /// <param name="ssmOnlyPolicyFile">Path to a policy file for SSM.</param>
    /// <param name="awsManagedPolicies">AWS Managed policies to be attached to the
role.</param>
    /// <returns>The Arn of the profile.</returns>
    public async Task<string> CreateInstanceProfileWithName(
        string policyName,
        string roleName,
        string profileName,
        string ssmOnlyPolicyFile,
        List<string>? awsManagedPolicies = null)
    {

        var assumeRoleDoc = "{" +
                                    "\"Version\": \"2012-10-17\"," +
                                    "\"Statement\": [{" +
                                         "\"Effect\": \"Allow\"," +
                                         "\"Principal\": {" +
                                         "\"Service\": [" +
                                             "\"ec2.amazonaws.com\"" +
                                         "]" +
                                         "}," +
                                    "\"Action\": \"sts:AssumeRole\"" +
                                    "}]" +
                                "}";

        var policyDocument = await File.ReadAllTextAsync(ssmOnlyPolicyFile);

        var policyArn = "";

        try
        {
            var createPolicyResult = await _amazonIam.CreatePolicyAsync(
                new CreatePolicyRequest
                {
                    PolicyName = policyName,
                    PolicyDocument = policyDocument
                });
            policyArn = createPolicyResult.Policy.Arn;
        }
        catch (EntityAlreadyExistsException)
```

```
        {
            // The policy already exists, so we look it up to get the Arn.
            var policiesPaginator = _amazonIam.Paginators.ListPolicies(
                new ListPoliciesRequest()
                {
                    Scope = PolicyScopeType.Local
                });
            // Get the entire list using the paginator.
            await foreach (var policy in policiesPaginator.Policies)
            {
                if (policy.PolicyName.Equals(policyName))
                {
                    policyArn = policy.Arn;
                }
            }

            if (policyArn == null)
            {
                throw new InvalidOperationException("Policy not found");
            }
        }

        try
        {
            await _amazonIam.CreateRoleAsync(new CreateRoleRequest()
            {
                RoleName = roleName,
                AssumeRolePolicyDocument = assumeRoleDoc,
            });
            await _amazonIam.AttachRolePolicyAsync(new AttachRolePolicyRequest()
            {
                RoleName = roleName,
                PolicyArn = policyArn
            });
            if (awsManagedPolicies != null)
            {
                foreach (var awsPolicy in awsManagedPolicies)
                {
                    await _amazonIam.AttachRolePolicyAsync(new
AttachRolePolicyRequest()
                    {
                        PolicyArn = $"arn:aws:iam::aws:policy/{awsPolicy}",
                        RoleName = roleName
                    });
```

```
                }
            }
        }
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine("Role already exists.");
        }

        string profileArn = "";
        try
        {
            var profileCreateResponse = await _amazonIam.CreateInstanceProfileAsync(
                new CreateInstanceProfileRequest()
                {
                    InstanceProfileName = profileName
                });
            // Allow time for the profile to be ready.
            profileArn = profileCreateResponse.InstanceProfile.Arn;
            Thread.Sleep(10000);
            await _amazonIam.AddRoleToInstanceProfileAsync(
                new AddRoleToInstanceProfileRequest()
                {
                    InstanceProfileName = profileName,
                    RoleName = roleName
                });

        }
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine("Policy already exists.");
            var profileGetResponse = await _amazonIam.GetInstanceProfileAsync(
                new GetInstanceProfileRequest()
                {
                    InstanceProfileName = profileName
                });
            profileArn = profileGetResponse.InstanceProfile.Arn;
        }
        return profileArn;
    }

    /// <summary>
    /// Create a new key pair and save the file.
    /// </summary>
    /// <param name="newKeyPairName">The name of the new key pair.</param>
```

```
    /// <returns>Async task.</returns>
    public async Task CreateKeyPair(string newKeyPairName)
    {
        try
        {
            var keyResponse = await _amazonEc2.CreateKeyPairAsync(
                new CreateKeyPairRequest() { KeyName = newKeyPairName });
            await File.WriteAllTextAsync($"{newKeyPairName}.pem",
                keyResponse.KeyPair.KeyMaterial);
            Console.WriteLine($"Created key pair {newKeyPairName}.");
        }
        catch (AlreadyExistsException)
        {
            Console.WriteLine("Key pair already exists.");
        }
    }

    /// <summary>
    /// Delete the key pair and file by name.
    /// </summary>
    /// <param name="deleteKeyPairName">The key pair to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteKeyPairByName(string deleteKeyPairName)
    {
        try
        {
            await _amazonEc2.DeleteKeyPairAsync(
                new DeleteKeyPairRequest() { KeyName = deleteKeyPairName });
            File.Delete($"{deleteKeyPairName}.pem");
        }
        catch (FileNotFoundException)
        {
            Console.WriteLine($"Key pair {deleteKeyPairName} not found.");
        }
    }

    /// <summary>
    /// Creates an Amazon EC2 launch template to use with Amazon EC2 Auto Scaling.
    /// The launch template specifies a Bash script in its user data field that runs
after
    /// the instance is started. This script installs the Python packages and starts
a Python
    /// web server on the instance.
    /// </summary>
```

```
   /// <param name="startupScriptPath">The path to a Bash script file that is
run.</param>
   /// <param name="instancePolicyPath">The path to a permissions policy to create
and attach to the profile.</param>
   /// <returns>The template object.</returns>
   public async Task<Amazon.EC2.Model.LaunchTemplate> CreateTemplate(string
startupScriptPath, string instancePolicyPath)
   {
       await CreateKeyPair(_keyPairName);
       await CreateInstanceProfileWithName(_instancePolicyName, _instanceRoleName,
_instanceProfileName, instancePolicyPath);

       var startServerText = await File.ReadAllTextAsync(startupScriptPath);
       var plainTextBytes = System.Text.Encoding.UTF8.GetBytes(startServerText);

       var amiLatest = await _amazonSsm.GetParameterAsync(
           new GetParameterRequest() { Name = _amiParam });
       var amiId = amiLatest.Parameter.Value;
       var launchTemplateResponse = await _amazonEc2.CreateLaunchTemplateAsync(
           new CreateLaunchTemplateRequest()
           {
               LaunchTemplateName = _launchTemplateName,
               LaunchTemplateData = new RequestLaunchTemplateData()
               {
                   InstanceType = _instanceType,
                   ImageId = amiId,
                   IamInstanceProfile =
                       new
                           LaunchTemplateIamInstanceProfileSpecificationRequest()
                       {
                           Name = _instanceProfileName
                       },
                   KeyName = _keyPairName,
                   UserData = System.Convert.ToBase64String(plainTextBytes)
               }
           });
       return launchTemplateResponse.LaunchTemplate;

   }

   /// <summary>
   /// Get a list of Availability Zones in the AWS Region of the Amazon EC2 Client.
   /// </summary>
   /// <returns>A list of availability zones.</returns>
```

```
    public async Task<List<string>> DescribeAvailabilityZones()
    {
        var zoneResponse = await _amazonEc2.DescribeAvailabilityZonesAsync(
            new DescribeAvailabilityZonesRequest());
        return zoneResponse.AvailabilityZones.Select(z => z.ZoneName).ToList();
    }

    /// <summary>
    /// Create an EC2 Auto Scaling group of a specified size and name.
    /// </summary>
    /// <param name="groupSize">The size for the group.</param>
    /// <param name="groupName">The name for the group.</param>
    /// <param name="availabilityZones">The availability zones for the group.</
param>
    /// <returns>Async task.</returns>
    public async Task CreateGroupOfSize(int groupSize, string groupName,
 List<string> availabilityZones)
    {
        try
        {
            await _amazonAutoScaling.CreateAutoScalingGroupAsync(
                new CreateAutoScalingGroupRequest()
                {
                    AutoScalingGroupName = groupName,
                    AvailabilityZones = availabilityZones,
                    LaunchTemplate =
                        new Amazon.AutoScaling.Model.LaunchTemplateSpecification()
                        {
                            LaunchTemplateName = _launchTemplateName,
                            Version = "$Default"
                        },
                    MaxSize = groupSize,
                    MinSize = groupSize
                });
            Console.WriteLine($"Created EC2 Auto Scaling group {groupName} with size
 {groupSize}.");
        }
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine($"EC2 Auto Scaling group {groupName} already
 exists.");
        }
    }
```

```
    /// <summary>
    /// Get the default VPC for the account.
    /// </summary>
    /// <returns>The default VPC object.</returns>
    public async Task<Vpc> GetDefaultVpc()
    {
        var vpcResponse = await _amazonEc2.DescribeVpcsAsync(
            new DescribeVpcsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("is-default", new List<string>() { "true" })
                }
            });
        return vpcResponse.Vpcs[0];
    }

    /// <summary>
    /// Get all the subnets for a Vpc in a set of availability zones.
    /// </summary>
    /// <param name="vpcId">The Id of the Vpc.</param>
    /// <param name="availabilityZones">The list of availability zones.</param>
    /// <returns>The collection of subnet objects.</returns>
    public async Task<List<Subnet>> GetAllVpcSubnetsForZones(string vpcId,
List<string> availabilityZones)
    {
        var subnets = new List<Subnet>();
        var subnetPaginator = _amazonEc2.Paginators.DescribeSubnets(
            new DescribeSubnetsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("vpc-id", new List<string>() { vpcId}),
                    new ("availability-zone", availabilityZones),
                    new ("default-for-az", new List<string>() { "true" })
                }
            });

        // Get the entire list using the paginator.
        await foreach (var subnet in subnetPaginator.Subnets)
        {
            subnets.Add(subnet);
        }
```

```
        return subnets;
    }

    /// <summary>
    /// Delete a launch template by name.
    /// </summary>
    /// <param name="templateName">The name of the template to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteTemplateByName(string templateName)
    {
        try
        {
            await _amazonEc2.DeleteLaunchTemplateAsync(
                new DeleteLaunchTemplateRequest()
                {
                    LaunchTemplateName = templateName
                });
        }
        catch (AmazonClientException)
        {
            Console.WriteLine($"Unable to delete template {templateName}.");
        }
    }

    /// <summary>
    /// Detaches a role from an instance profile, detaches policies from the role,
    /// and deletes all the resources.
    /// </summary>
    /// <param name="profileName">The name of the profile to delete.</param>
    /// <param name="roleName">The name of the role to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteInstanceProfile(string profileName, string roleName)
    {
        try
        {
            await _amazonIam.RemoveRoleFromInstanceProfileAsync(
                new RemoveRoleFromInstanceProfileRequest()
                {
                    InstanceProfileName = profileName,
                    RoleName = roleName
                });
            await _amazonIam.DeleteInstanceProfileAsync(
                new DeleteInstanceProfileRequest() { InstanceProfileName =
profileName });
```

```
            var attachedPolicies = await _amazonIam.ListAttachedRolePoliciesAsync(
                new ListAttachedRolePoliciesRequest() { RoleName = roleName });
            foreach (var policy in attachedPolicies.AttachedPolicies)
            {
                await _amazonIam.DetachRolePolicyAsync(
                    new DetachRolePolicyRequest()
                    {
                        RoleName = roleName,
                        PolicyArn = policy.PolicyArn
                    });
                // Delete the custom policies only.
                if (!policy.PolicyArn.StartsWith("arn:aws:iam::aws"))
                {
                    await _amazonIam.DeletePolicyAsync(
                        new Amazon.IdentityManagement.Model.DeletePolicyRequest()
                        {
                            PolicyArn = policy.PolicyArn
                        });
                }
            }

            await _amazonIam.DeleteRoleAsync(
                new DeleteRoleRequest() { RoleName = roleName });
        }
        catch (NoSuchEntityException)
        {
            Console.WriteLine($"Instance profile {profileName} does not exist.");
        }
    }

    /// <summary>
    /// Gets data about the instances in an EC2 Auto Scaling group by its group
name.
    /// </summary>
    /// <param name="group">The name of the auto scaling group.</param>
    /// <returns>A collection of instance Ids.</returns>
    public async Task<IEnumerable<string>> GetInstancesByGroupName(string group)
    {
        var instanceResponse = await
_amazonAutoScaling.DescribeAutoScalingGroupsAsync(
            new DescribeAutoScalingGroupsRequest()
            {
                AutoScalingGroupNames = new List<string>() { group }
            });
```

```
        var instanceIds = instanceResponse.AutoScalingGroups.SelectMany(
            g => g.Instances.Select(i => i.InstanceId));
        return instanceIds;
    }

    /// <summary>
    /// Get the instance profile association data for an instance.
    /// </summary>
    /// <param name="instanceId">The Id of the instance.</param>
    /// <returns>Instance profile associations data.</returns>
    public async Task<IamInstanceProfileAssociation> GetInstanceProfile(string
instanceId)
    {
        var response = await _amazonEc2.DescribeIamInstanceProfileAssociationsAsync(
            new DescribeIamInstanceProfileAssociationsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("instance-id", new List<string>() { instanceId })
                },
            });
        return response.IamInstanceProfileAssociations[0];
    }

    /// <summary>
    /// Replace the profile associated with a running instance. After the profile is
replaced, the instance
    /// is rebooted to ensure that it uses the new profile. When the instance is
ready, Systems Manager is
    /// used to restart the Python web server.
    /// </summary>
    /// <param name="instanceId">The Id of the instance to update.</param>
    /// <param name="credsProfileName">The name of the new profile to associate with
the specified instance.</param>
    /// <param name="associationId">The Id of the existing profile association for
the instance.</param>
    /// <returns>Async task.</returns>
    public async Task ReplaceInstanceProfile(string instanceId, string
credsProfileName, string associationId)
    {
        await _amazonEc2.ReplaceIamInstanceProfileAssociationAsync(
                new ReplaceIamInstanceProfileAssociationRequest()
                {
                    AssociationId = associationId,
```

```
                    IamInstanceProfile = new IamInstanceProfileSpecification()
                    {
                        Name = credsProfileName
                    }
                });
        // Allow time before resetting.
        Thread.Sleep(25000);
        var instanceReady = false;
        var retries = 5;
        while (retries-- > 0 && !instanceReady)
        {
            await _amazonEc2.RebootInstancesAsync(
                new RebootInstancesRequest(new List<string>() { instanceId }));
            Thread.Sleep(10000);

            var instancesPaginator =
_amazonSsm.Paginators.DescribeInstanceInformation(
                new DescribeInstanceInformationRequest());
            // Get the entire list using the paginator.
            await foreach (var instance in
instancesPaginator.InstanceInformationList)
            {
                instanceReady = instance.InstanceId == instanceId;
                if (instanceReady)
                {
                    break;
                }
            }
        }
        Console.WriteLine($"Sending restart command to instance {instanceId}");
        await _amazonSsm.SendCommandAsync(
            new SendCommandRequest()
            {
                InstanceIds = new List<string>() { instanceId },
                DocumentName = "AWS-RunShellScript",
                Parameters = new Dictionary<string, List<string>>()
                {
                        {"commands", new List<string>() { "cd / && sudo python3
server.py 80" }}
                }
            });
        Console.WriteLine($"Restarted the web server on instance {instanceId}");
    }
```

```
    /// <summary>
    /// Try to terminate an instance by its Id.
    /// </summary>
    /// <param name="instanceId">The Id of the instance to terminate.</param>
    /// <returns>Async task.</returns>
    public async Task TryTerminateInstanceById(string instanceId)
    {
        var stopping = false;
        Console.WriteLine($"Stopping {instanceId}...");
        while (!stopping)
        {
            try
            {
                await _amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(
                    new TerminateInstanceInAutoScalingGroupRequest()
                    {
                        InstanceId = instanceId,
                        ShouldDecrementDesiredCapacity = false
                    });
                stopping = true;
            }
            catch (ScalingActivityInProgressException)
            {
                Console.WriteLine($"Scaling activity in progress for {instanceId}.
Waiting...");
                Thread.Sleep(10000);
            }
        }
    }

    /// <summary>
    /// Tries to delete the EC2 Auto Scaling group. If the group is in use or in
progress,
    /// waits and retries until the group is successfully deleted.
    /// </summary>
    /// <param name="groupName">The name of the group to try to delete.</param>
    /// <returns>Async task.</returns>
    public async Task TryDeleteGroupByName(string groupName)
    {
        var stopped = false;
        while (!stopped)
        {
            try
            {
```

```
                await _amazonAutoScaling.DeleteAutoScalingGroupAsync(
                    new DeleteAutoScalingGroupRequest()
                    {
                        AutoScalingGroupName = groupName
                    });
                stopped = true;
            }
            catch (Exception e)
                when ((e is ScalingActivityInProgressException)
                      || (e is Amazon.AutoScaling.Model.ResourceInUseException))
            {
                Console.WriteLine($"Some instances are still running. Waiting...");
                Thread.Sleep(10000);
            }
        }
    }

    /// <summary>
    /// Terminate instances and delete the Auto Scaling group by name.
    /// </summary>
    /// <param name="groupName">The name of the group to delete.</param>
    /// <returns>Async task.</returns>
    public async Task TerminateAndDeleteAutoScalingGroupWithName(string groupName)
    {
        var describeGroupsResponse = await
_amazonAutoScaling.DescribeAutoScalingGroupsAsync(
            new DescribeAutoScalingGroupsRequest()
            {
                AutoScalingGroupNames = new List<string>() { groupName }
            });
        if (describeGroupsResponse.AutoScalingGroups.Any())
        {
            // Update the size to 0.
            await _amazonAutoScaling.UpdateAutoScalingGroupAsync(
                new UpdateAutoScalingGroupRequest()
                {
                    AutoScalingGroupName = groupName,
                    MinSize = 0
                });
            var group = describeGroupsResponse.AutoScalingGroups[0];
            foreach (var instance in group.Instances)
            {
                await TryTerminateInstanceById(instance.InstanceId);
            }
```

```
            await TryDeleteGroupByName(groupName);
        }
        else
        {
            Console.WriteLine($"No groups found with name {groupName}.");
        }
    }


    /// <summary>
    /// Get the default security group for a specified Vpc.
    /// </summary>
    /// <param name="vpc">The Vpc to search.</param>
    /// <returns>The default security group.</returns>
    public async Task<SecurityGroup> GetDefaultSecurityGroupForVpc(Vpc vpc)
    {
        var groupResponse = await _amazonEc2.DescribeSecurityGroupsAsync(
            new DescribeSecurityGroupsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("group-name", new List<string>() { "default" }),
                    new ("vpc-id", new List<string>() { vpc.VpcId })
                }
            });
        return groupResponse.SecurityGroups[0];
    }

    /// <summary>
    /// Verify the default security group of a Vpc allows ingress from the calling
computer.
    /// This can be done by allowing ingress from this computer's IP address.
    /// In some situations, such as connecting from a corporate network, you must
instead specify
    /// a prefix list Id. You can also temporarily open the port to any IP address
while running this example.
    /// If you do, be sure to remove public access when you're done.
    /// </summary>
    /// <param name="vpc">The group to check.</param>
    /// <param name="port">The port to verify.</param>
    /// <param name="ipAddress">This computer's IP address.</param>
    /// <returns>True if the ip address is allowed on the group.</returns>
```

```
   public bool VerifyInboundPortForGroup(SecurityGroup group, int port, string
ipAddress)
   {
       var portIsOpen = false;
       foreach (var ipPermission in group.IpPermissions)
       {
           if (ipPermission.FromPort == port)
           {
               foreach (var ipRange in ipPermission.Ipv4Ranges)
               {
                   var cidr = ipRange.CidrIp;
                   if (cidr.StartsWith(ipAddress) || cidr == "0.0.0.0/0")
                   {
                       portIsOpen = true;
                   }
               }

               if (ipPermission.PrefixListIds.Any())
               {
                   portIsOpen = true;
               }

               if (!portIsOpen)
               {
                   Console.WriteLine("The inbound rule does not appear to be open
to either this computer's IP\n" +
                                     "address, to all IP addresses (0.0.0.0/0), or
to a prefix list ID.");
               }
               else
               {
                   break;
               }
           }
       }

       return portIsOpen;
   }

   /// <summary>
   /// Add an ingress rule to the specified security group that allows access on
the
   /// specified port from the specified IP address.
   /// </summary>
```

```
    /// <param name="groupId">The Id of the security group to modify.</param>
    /// <param name="port">The port to open.</param>
    /// <param name="ipAddress">The IP address to allow access.</param>
    /// <returns>Async task.</returns>
    public async Task OpenInboundPort(string groupId, int port, string ipAddress)
    {
        await _amazonEc2.AuthorizeSecurityGroupIngressAsync(
            new AuthorizeSecurityGroupIngressRequest()
            {
                GroupId = groupId,
                IpPermissions = new List<IpPermission>()
                {
                    new IpPermission()
                    {
                        FromPort = port,
                        ToPort = port,
                        IpProtocol = "tcp",
                        Ipv4Ranges = new List<IpRange>()
                        {
                            new IpRange() { CidrIp = $"{ipAddress}/32" }
                        }
                    }
                }
            });
    }

    /// <summary>
    /// Attaches an Elastic Load Balancing (ELB) target group to this EC2 Auto
 Scaling group.
    /// The
    /// </summary>
    /// <param name="autoScalingGroupName">The name of the Auto Scaling group.</
param>
    /// <param name="targetGroupArn">The Arn for the target group.</param>
    /// <returns>Async task.</returns>
    public async Task AttachLoadBalancerToGroup(string autoScalingGroupName, string
 targetGroupArn)
    {
        await _amazonAutoScaling.AttachLoadBalancerTargetGroupsAsync(
            new AttachLoadBalancerTargetGroupsRequest()
            {
                AutoScalingGroupName = autoScalingGroupName,
                TargetGroupARNs = new List<string>() { targetGroupArn }
            });
```

```
    }
}
```

Elastic Load Balancing のアクションをラップするクラスを作成します。

```
/// <summary>
/// Encapsulates Elastic Load Balancer actions.
/// </summary>
public class ElasticLoadBalancerWrapper
{
    private readonly IAmazonElasticLoadBalancingV2 _amazonElasticLoadBalancingV2;
    private string? _endpoint = null;
    private readonly string _targetGroupName = "";
    private readonly string _loadBalancerName = "";
    HttpClient _httpClient = new();

    public string TargetGroupName => _targetGroupName;
    public string LoadBalancerName => _loadBalancerName;

    /// <summary>
    /// Constructor for the Elastic Load Balancer wrapper.
    /// </summary>
    /// <param name="amazonElasticLoadBalancingV2">The injected load balancing v2
 client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public ElasticLoadBalancerWrapper(
        IAmazonElasticLoadBalancingV2 amazonElasticLoadBalancingV2,
        IConfiguration configuration)
    {
        _amazonElasticLoadBalancingV2 = amazonElasticLoadBalancingV2;
        var prefix = configuration["resourcePrefix"];
        _targetGroupName = prefix + "-tg";
        _loadBalancerName = prefix + "-lb";
    }

    /// <summary>
    /// Get the HTTP Endpoint of a load balancer by its name.
    /// </summary>
    /// <param name="loadBalancerName">The name of the load balancer.</param>
    /// <returns>The HTTP endpoint.</returns>
```

```
    public async Task<string> GetEndpointForLoadBalancerByName(string
loadBalancerName)
    {
        if (_endpoint == null)
        {
            var endpointResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                    new DescribeLoadBalancersRequest()
                    {
                        Names = new List<string>() { loadBalancerName }
                    });
            _endpoint = endpointResponse.LoadBalancers[0].DNSName;
        }

        return _endpoint;
    }

    /// <summary>
    /// Return the GET response for an endpoint as text.
    /// </summary>
    /// <param name="endpoint">The endpoint for the request.</param>
    /// <returns>The request response.</returns>
    public async Task<string> GetEndPointResponse(string endpoint)
    {
        var endpointResponse = await _httpClient.GetAsync($"http://{endpoint}");
        var textResponse = await endpointResponse.Content.ReadAsStringAsync();
        return textResponse!;
    }

    /// <summary>
    /// Get the target health for a group by name.
    /// </summary>
    /// <param name="groupName">The name of the group.</param>
    /// <returns>The collection of health descriptions.</returns>
    public async Task<List<TargetHealthDescription>>
CheckTargetHealthForGroup(string groupName)
    {
        List<TargetHealthDescription> result = null!;
        try
        {
            var groupResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                    new DescribeTargetGroupsRequest()
                    {
```

```
                        Names = new List<string>() { groupName }
                    });
            var healthResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetHealthAsync(
                    new DescribeTargetHealthRequest()
                    {
                        TargetGroupArn =
 groupResponse.TargetGroups[0].TargetGroupArn
                    });
            ;
            result = healthResponse.TargetHealthDescriptions;
        }
        catch (TargetGroupNotFoundException)
        {
            Console.WriteLine($"Target group {groupName} not found.");
        }
        return result;
    }

    /// <summary>
    /// Create an Elastic Load Balancing target group. The target group specifies
 how the load balancer forwards
    /// requests to instances in the group and how instance health is checked.
    ///
    /// To speed up this demo, the health check is configured with shortened times
 and lower thresholds. In production,
    /// you might want to decrease the sensitivity of your health checks to avoid
 unwanted failures.
    /// </summary>
    /// <param name="groupName">The name for the group.</param>
    /// <param name="protocol">The protocol, such as HTTP.</param>
    /// <param name="port">The port to use to forward requests, such as 80.</param>
    /// <param name="vpcId">The Id of the Vpc in which the load balancer exists.</
param>
    /// <returns>The new TargetGroup object.</returns>
    public async Task<TargetGroup> CreateTargetGroupOnVpc(string groupName,
 ProtocolEnum protocol, int port, string vpcId)
    {
        var createResponse = await
 _amazonElasticLoadBalancingV2.CreateTargetGroupAsync(
            new CreateTargetGroupRequest()
            {
                Name = groupName,
                Protocol = protocol,
```

```
                Port = port,
                HealthCheckPath = "/healthcheck",
                HealthCheckIntervalSeconds = 10,
                HealthCheckTimeoutSeconds = 5,
                HealthyThresholdCount = 2,
                UnhealthyThresholdCount = 2,
                VpcId = vpcId
            });
        var targetGroup = createResponse.TargetGroups[0];
        return targetGroup;
    }

    /// <summary>
    /// Create an Elastic Load Balancing load balancer that uses the specified
subnets
    /// and forwards requests to the specified target group.
    /// </summary>
    /// <param name="name">The name for the new load balancer.</param>
    /// <param name="subnetIds">Subnets for the load balancer.</param>
    /// <param name="targetGroup">Target group for forwarded requests.</param>
    /// <returns>The new LoadBalancer object.</returns>
    public async Task<LoadBalancer> CreateLoadBalancerAndListener(string name,
List<string> subnetIds, TargetGroup targetGroup)
    {
        var createLbResponse = await
_amazonElasticLoadBalancingV2.CreateLoadBalancerAsync(
            new CreateLoadBalancerRequest()
            {
                Name = name,
                Subnets = subnetIds
            });
        var loadBalancerArn = createLbResponse.LoadBalancers[0].LoadBalancerArn;

        // Wait for load balancer to be available.
        var loadBalancerReady = false;
        while (!loadBalancerReady)
        {
            try
            {
                var describeResponse =
                    await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                        new DescribeLoadBalancersRequest()
                        {
                            Names = new List<string>() { name }
```

```
                        });

                var loadBalancerState =
describeResponse.LoadBalancers[0].State.Code;

                loadBalancerReady = loadBalancerState ==
LoadBalancerStateEnum.Active;
            }
            catch (LoadBalancerNotFoundException)
            {
                loadBalancerReady = false;
            }
            Thread.Sleep(10000);
        }
        // Create the listener.
        await _amazonElasticLoadBalancingV2.CreateListenerAsync(
            new CreateListenerRequest()
            {
                LoadBalancerArn = loadBalancerArn,
                Protocol = targetGroup.Protocol,
                Port = targetGroup.Port,
                DefaultActions = new List<Action>()
                {
                    new Action()
                    {
                        Type = ActionTypeEnum.Forward,
                        TargetGroupArn = targetGroup.TargetGroupArn
                    }
                }
            });
        return createLbResponse.LoadBalancers[0];
    }

    /// <summary>
    /// Verify this computer can successfully send a GET request to the
    /// load balancer endpoint.
    /// </summary>
    /// <param name="endpoint">The endpoint to check.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> VerifyLoadBalancerEndpoint(string endpoint)
    {
        var success = false;
        var retries = 3;
        while (!success && retries > 0)
```

```
        {
            try
            {
                var endpointResponse = await _httpClient.GetAsync($"http://
{endpoint}");
                Console.WriteLine($"Response: {endpointResponse.StatusCode}.");

                if (endpointResponse.IsSuccessStatusCode)
                {
                    success = true;
                }
                else
                {
                    retries = 0;
                }
            }
            catch (HttpRequestException)
            {
                Console.WriteLine("Connection error, retrying...");
                retries--;
                Thread.Sleep(10000);
            }
        }

        return success;
    }

    /// <summary>
    /// Delete a load balancer by its specified name.
    /// </summary>
    /// <param name="name">The name of the load balancer to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteLoadBalancerByName(string name)
    {
        try
        {
            var describeLoadBalancerResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                    new DescribeLoadBalancersRequest()
                    {
                        Names = new List<string>() { name }
                    });
            var lbArn =
 describeLoadBalancerResponse.LoadBalancers[0].LoadBalancerArn;
```

```
            await _amazonElasticLoadBalancingV2.DeleteLoadBalancerAsync(
                new DeleteLoadBalancerRequest()
                {
                    LoadBalancerArn = lbArn
                }
            );
        }
        catch (LoadBalancerNotFoundException)
        {
            Console.WriteLine($"Load balancer {name} not found.");
        }
    }

    /// <summary>
    /// Delete a TargetGroup by its specified name.
    /// </summary>
    /// <param name="groupName">Name of the group to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteTargetGroupByName(string groupName)
    {
        var done = false;
        while (!done)
        {
            try
            {
                var groupResponse =
                    await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                        new DescribeTargetGroupsRequest()
                        {
                            Names = new List<string>() { groupName }
                        });

                var targetArn = groupResponse.TargetGroups[0].TargetGroupArn;
                await _amazonElasticLoadBalancingV2.DeleteTargetGroupAsync(
                    new DeleteTargetGroupRequest() { TargetGroupArn = targetArn });
                Console.WriteLine($"Deleted load balancing target group
{groupName}.");
                done = true;
            }
            catch (TargetGroupNotFoundException)
            {
                Console.WriteLine(
                    $"Target group {groupName} not found, could not delete.");
                done = true;
```

```
            }
            catch (ResourceInUseException)
            {
                Console.WriteLine("Target group not yet released, waiting...");
                Thread.Sleep(10000);
            }
        }
    }
}
```

DynamoDB を使用してレコメンデーションサービスをシミュレートするクラスを作成します。

```
/// <summary>
/// Encapsulates a DynamoDB table to use as a service that recommends books, movies,
 and songs.
/// </summary>
public class Recommendations
{
    private readonly IAmazonDynamoDB _amazonDynamoDb;
    private readonly DynamoDBContext _context;
    private readonly string _tableName;

    public string TableName => _tableName;

    /// <summary>
    /// Constructor for the Recommendations service.
    /// </summary>
    /// <param name="amazonDynamoDb">The injected DynamoDb client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public Recommendations(IAmazonDynamoDB amazonDynamoDb, IConfiguration
 configuration)
    {
        _amazonDynamoDb = amazonDynamoDb;
        _context = new DynamoDBContext(_amazonDynamoDb);
        _tableName = configuration["databaseName"]!;
    }

    /// <summary>
    /// Create the DynamoDb table with a specified name.
    /// </summary>
    /// <param name="tableName">The name for the table.</param>
    /// <returns>True when ready.</returns>
```

```
public async Task<bool> CreateDatabaseWithName(string tableName)
{
    try
    {
        Console.Write($"Creating table {tableName}...");
        var createRequest = new CreateTableRequest()
        {
            TableName = tableName,
            AttributeDefinitions = new List<AttributeDefinition>()
                {
                    new AttributeDefinition()
                    {
                        AttributeName = "MediaType",
                        AttributeType = ScalarAttributeType.S
                    },
                    new AttributeDefinition()
                    {
                        AttributeName = "ItemId",
                        AttributeType = ScalarAttributeType.N
                    }
                },
            KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement()
                    {
                        AttributeName = "MediaType",
                        KeyType = KeyType.HASH
                    },
                    new KeySchemaElement()
                    {
                        AttributeName = "ItemId",
                        KeyType = KeyType.RANGE
                    }
                },
            ProvisionedThroughput = new ProvisionedThroughput()
            {
                ReadCapacityUnits = 5,
                WriteCapacityUnits = 5
            }
        };
        await _amazonDynamoDb.CreateTableAsync(createRequest);

        // Wait until the table is ACTIVE and then report success.
        Console.Write("\nWaiting for table to become active...");
```

```
            var request = new DescribeTableRequest
            {
                TableName = tableName
            };

            TableStatus status;
            do
            {
                Thread.Sleep(2000);

                var describeTableResponse = await
 _amazonDynamoDb.DescribeTableAsync(request);
                status = describeTableResponse.Table.TableStatus;

                Console.Write(".");
            }
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException)
        {
            Console.WriteLine($"Table {tableName} already exists.");
            return false;
        }
    }

    /// <summary>
    /// Populate the database table with data from a specified path.
    /// </summary>
    /// <param name="databaseTableName">The name of the table.</param>
    /// <param name="recommendationsPath">The path of the recommendations data.</
param>
    /// <returns>Async task.</returns>
    public async Task PopulateDatabase(string databaseTableName, string
 recommendationsPath)
    {
        var recommendationsText = await File.ReadAllTextAsync(recommendationsPath);
        var records =
            JsonSerializer.Deserialize<RecommendationModel[]>(recommendationsText);
        var batchWrite = _context.CreateBatchWrite<RecommendationModel>();

        foreach (var record in records!)
```

```
        {
            batchWrite.AddPutItem(record);
        }

        await batchWrite.ExecuteAsync();
    }

    /// <summary>
    /// Delete the recommendation table by name.
    /// </summary>
    /// <param name="tableName">The name of the recommendation table.</param>
    /// <returns>Async task.</returns>
    public async Task DestroyDatabaseByName(string tableName)
    {
        try
        {
            await _amazonDynamoDb.DeleteTableAsync(
                new DeleteTableRequest() { TableName = tableName });
            Console.WriteLine($"Table {tableName} was deleted.");
        }
        catch (ResourceNotFoundException)
        {
            Console.WriteLine($"Table {tableName} not found");
        }
    }
}
```

Systems Manager のアクションをラップするクラスを作成します。

```
/// <summary>
/// Encapsulates Systems Manager parameter operations. This example uses these
 parameters
/// to drive the demonstration of resilient architecture, such as failure of a
 dependency or
/// how the service responds to a health check.
/// </summary>
public class SmParameterWrapper
{
    private readonly IAmazonSimpleSystemsManagement _amazonSimpleSystemsManagement;

    private readonly string _tableParameter = "doc-example-resilient-architecture-
table";
```

```
    private readonly string _failureResponseParameter = "doc-example-resilient-
architecture-failure-response";
    private readonly string _healthCheckParameter = "doc-example-resilient-
architecture-health-check";
    private readonly string _tableName = "";

    public string TableParameter => _tableParameter;
    public string TableName => _tableName;
    public string HealthCheckParameter => _healthCheckParameter;
    public string FailureResponseParameter => _failureResponseParameter;

    /// <summary>
    /// Constructor for the SmParameterWrapper.
    /// </summary>
    /// <param name="amazonSimpleSystemsManagement">The injected Simple Systems
 Management client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public SmParameterWrapper(IAmazonSimpleSystemsManagement
 amazonSimpleSystemsManagement, IConfiguration configuration)
    {
        _amazonSimpleSystemsManagement = amazonSimpleSystemsManagement;
        _tableName = configuration["databaseName"]!;
    }

    /// <summary>
    /// Reset the Systems Manager parameters to starting values for the demo.
    /// </summary>
    /// <returns>Async task.</returns>
    public async Task Reset()
    {
        await this.PutParameterByName(_tableParameter, _tableName);
        await this.PutParameterByName(_failureResponseParameter, "none");
        await this.PutParameterByName(_healthCheckParameter, "shallow");
    }

    /// <summary>
    /// Set the value of a named Systems Manager parameter.
    /// </summary>
    /// <param name="name">The name of the parameter.</param>
    /// <param name="value">The value to set.</param>
    /// <returns>Async task.</returns>
    public async Task PutParameterByName(string name, string value)
    {
        await _amazonSimpleSystemsManagement.PutParameterAsync(
```

```
            new PutParameterRequest() { Name = name, Value = value, Overwrite =
 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。
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer
  - CreateTargetGroup
  - DeleteAutoScalingGroup
  - DeleteInstanceProfile
  - DeleteLaunchTemplate
  - DeleteLoadBalancer
  - DeleteTargetGroup
  - DescribeAutoScalingGroups
  - DescribeAvailabilityZones
  - DescribeIamInstanceProfileAssociations
  - DescribeInstances
  - DescribeLoadBalancers
  - DescribeSubnets
  - DescribeTargetGroups
  - DescribeTargetHealth
  - DescribeVpcs
  - RebootInstances
  - ReplaceIamInstanceProfileAssociation
  - TerminateInstanceInAutoScalingGroup

- UpdateAutoScalingGroup

EventBridge を使用した の例 AWS SDK for .NET

次のコード例は、 AWS SDK for .NET で 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 EventBridge。

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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。このリンクには GitHub、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が記載されています。

開始方法

こんにち EventBridgeは

次のコード例は、 の使用を開始する方法を示しています EventBridge。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
using Amazon.EventBridge;
using Amazon.EventBridge.Model;

namespace EventBridgeActions;

public static class HelloEventBridge
{
```

```
    static async Task Main(string[] args)
    {
        var eventBridgeClient = new AmazonEventBridgeClient();

        Console.WriteLine($"Hello Amazon EventBridge! Following are some of your
 EventBuses:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get the first five event buses.
        var response = await eventBridgeClient.ListEventBusesAsync(
            new ListEventBusesRequest()
            {
                Limit = 5
            });

        foreach (var eventBus in response.EventBuses)
        {
            Console.WriteLine($"\tEventBus: {eventBus.Name}");
            Console.WriteLine($"\tArn: {eventBus.Arn}");
            Console.WriteLine($"\tPolicy: {eventBus.Policy}");
            Console.WriteLine();
        }
    }
}
```

- API の詳細については、「 API リファレンスListEventBuses」の「」を参照してください。AWS SDK for .NET

トピック

- アクション
- シナリオ

アクション

`DeleteRule`

次の例は、DeleteRule を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

その名前でルールを削除します。

```
/// <summary>
/// Delete an event rule by name.
/// </summary>
/// <param name="ruleName">The name of the event rule.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteRuleByName(string ruleName)
{
    var response = await _amazonEventBridge.DeleteRuleAsync(
        new DeleteRuleRequest()
        {
            Name = ruleName
        });

    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- API の詳細については、「 API リファレンスDeleteRule」の「」を参照してください。 AWS SDK for .NET

`DescribeRule`

次の例は、`DescribeRule` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ルールの説明を使用してルールの状態を取得します。

```
    /// <summary>
    /// Get the state for a rule by the rule name.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <param name="eventBusName">The optional name of the event bus. If empty,
uses the default event bus.</param>
    /// <returns>The state of the rule.</returns>
    public async Task<RuleState> GetRuleStateByRuleName(string ruleName, string?
eventBusName = null)
    {
        var ruleResponse = await _amazonEventBridge.DescribeRuleAsync(
            new DescribeRuleRequest()
            {
                Name = ruleName,
                EventBusName = eventBusName
            });
        return ruleResponse.State;
    }
```

- API の詳細については、「 API リファレンスDescribeRule」の「」を参照してください。 AWS SDK for .NET

`DisableRule`

次の例は、DisableRule を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

そのルール名でルールを無効化します。

```
    /// <summary>
    /// Disable a particular rule on an event bus.
```

```
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DisableRuleByName(string ruleName)
    {
        var ruleResponse = await _amazonEventBridge.DisableRuleAsync(
            new DisableRuleRequest()
            {
                Name = ruleName
            });
        return ruleResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスDisableRule」の「」を参照してください。 AWS SDK for .NET

`EnableRule`

次の例は、EnableRule を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

そのルール名でルールを有効化します。

```
    /// <summary>
    /// Enable a particular rule on an event bus.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> EnableRuleByName(string ruleName)
    {
        var ruleResponse = await _amazonEventBridge.EnableRuleAsync(
            new EnableRuleRequest()
            {
                Name = ruleName
```

```
            });
        return ruleResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスEnableRule」の「」を参照してください。 AWS SDK for .NET

`ListRuleNamesByTarget`

次の例は、ListRuleNamesByTarget を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ターゲットを使用してすべてのルール名を一覧表示します。

```
    /// <summary>
    /// List names of all rules matching a target.
    /// </summary>
    /// <param name="targetArn">The ARN of the target.</param>
    /// <returns>The list of rule names.</returns>
    public async Task<List<string>> ListAllRuleNamesByTarget(string targetArn)
    {
        var results = new List<string>();
        var request = new ListRuleNamesByTargetRequest()
        {
            TargetArn = targetArn
        };
        ListRuleNamesByTargetResponse response;
        do
        {
            response = await _amazonEventBridge.ListRuleNamesByTargetAsync(request);
            results.AddRange(response.RuleNames);
            request.NextToken = response.NextToken;

        } while (response.NextToken is not null);
```

```
        return results;
    }
```

- API の詳細については、「 API リファレンスListRuleNamesByTarget」の「」を参照してください。 AWS SDK for .NET

`ListRules`

次の例は、ListRules を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

イベントバスのルールをすべて一覧表示します。

```
    /// <summary>
    /// List the rules on an event bus.
    /// </summary>
    /// <param name="eventBusArn">The optional ARN of the event bus. If empty, uses
the default event bus.</param>
    /// <returns>The list of rules.</returns>
    public async Task<List<Rule>> ListAllRulesForEventBus(string? eventBusArn =
null)
    {
        var results = new List<Rule>();
        var request = new ListRulesRequest()
        {
            EventBusName = eventBusArn
        };
        // Get all of the pages of rules.
        ListRulesResponse response;
        do
        {
            response = await _amazonEventBridge.ListRulesAsync(request);
            results.AddRange(response.Rules);
```

```
            request.NextToken = response.NextToken;

        } while (response.NextToken is not null);

        return results;
    }
```

- API の詳細については、「 API リファレンスListRules」の「」を参照してください。 AWS SDK for .NET

`ListTargetsByRule`

次の例は、`ListTargetsByRule` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ルール名を使用してルールのすべてのターゲットを一覧表示します。

```
    /// <summary>
    /// List all of the targets matching a rule by name.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <returns>The list of targets.</returns>
    public async Task<List<Target>> ListAllTargetsOnRule(string ruleName)
    {
        var results = new List<Target>();
        var request = new ListTargetsByRuleRequest()
        {
            Rule = ruleName
        };
        ListTargetsByRuleResponse response;
        do
        {
            response = await _amazonEventBridge.ListTargetsByRuleAsync(request);
            results.AddRange(response.Targets);
```

```
            request.NextToken = response.NextToken;

        } while (response.NextToken is not null);

        return results;
    }
```

- API の詳細については、「 API リファレンスListTargetsByRule」の「」を参照してください。AWS SDK for .NET

`PutEvents`

次の例は、PutEvents を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ルールのカスタムパターンに一致するイベントを送信します。

```
    /// <summary>
    /// Add an event to the event bus that includes an email, message, and time.
    /// </summary>
    /// <param name="email">The email to use in the event detail of the custom
event.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutCustomEmailEvent(string email)
    {
        var eventDetail = new
        {
            UserEmail = email,
            Message = "This event was generated by example code.",
            UtcTime = DateTime.UtcNow.ToString("g")
        };
        var response = await _amazonEventBridge.PutEventsAsync(
            new PutEventsRequest()
            {
```

```
                Entries = new List<PutEventsRequestEntry>()
                {
                    new PutEventsRequestEntry()
                    {
                        Source = "ExampleSource",
                        Detail = JsonSerializer.Serialize(eventDetail),
                        DetailType = "ExampleType"
                    }
                }
            });

        return response.FailedEntryCount == 0;
    }
```

- API の詳細については、「 API リファレンスPutEvents」の「」を参照してください。 AWS SDK for .NET

`PutRule`

次の例は、`PutRule` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon Simple Storage Service バケットにオブジェクトが追加されたときにトリガーするルールを作成します。

```
    /// <summary>
    /// Create a new event rule that triggers when an Amazon S3 object is created in
 a bucket.
    /// </summary>
    /// <param name="roleArn">The ARN of the role.</param>
    /// <param name="ruleName">The name to give the rule.</param>
    /// <param name="bucketName">The name of the bucket to trigger the event.</
param>
```

```
    /// <returns>The ARN of the new rule.</returns>
    public async Task<string> PutS3UploadRule(string roleArn, string ruleName,
string bucketName)
    {
        string eventPattern = "{" +
                               "\"source\": [\"aws.s3\"]," +
                                   "\"detail-type\": [\"Object Created\"]," +
                                   "\"detail\": {" +
                                       "\"bucket\": {" +
                                           "\"name\": [\"" + bucketName + "\"]" +
                                       "}" +
                                   "}" +
                               "}";

        var response = await _amazonEventBridge.PutRuleAsync(
            new PutRuleRequest()
            {
                Name = ruleName,
                Description = "Example S3 upload rule for EventBridge",
                RoleArn = roleArn,
                EventPattern = eventPattern
            });

        return response.RuleArn;
    }
```

カスタムパターンを使用するルールを作成します。

```
    /// <summary>
    /// Update a rule to use a custom defined event pattern.
    /// </summary>
    /// <param name="ruleName">The name of the rule to update.</param>
    /// <returns>The ARN of the updated rule.</returns>
    public async Task<string> UpdateCustomEventPattern(string ruleName)
    {
        string customEventsPattern = "{" +
                                      "\"source\": [\"ExampleSource\"]," +
                                      "\"detail-type\": [\"ExampleType\"]" +
                                      "}";

        var response = await _amazonEventBridge.PutRuleAsync(
            new PutRuleRequest()
```

```
            {
                Name = ruleName,
                Description = "Custom test rule",
                EventPattern = customEventsPattern
            });

        return response.RuleArn;
    }
```

- API の詳細については、「 API リファレンスPutRule」の「」を参照してください。 AWS SDK for .NET

`PutTargets`

次の例は、`PutTargets` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon SNS トピックをルールのターゲットとして追加します。

```
    /// <summary>
    /// Add an Amazon SNS target topic to a rule.
    /// </summary>
    /// <param name="ruleName">The name of the rule to update.</param>
    /// <param name="targetArn">The ARN of the Amazon SNS target.</param>
    /// <param name="eventBusArn">The optional event bus name, uses default if
empty.</param>
    /// <returns>The ID of the target.</returns>
    public async Task<string> AddSnsTargetToRule(string ruleName, string targetArn,
string? eventBusArn = null)
    {
        var targetID = Guid.NewGuid().ToString();

        // Create the list of targets and add a new target.
        var targets = new List<Target>
```

```
        {
            new Target()
            {
                Arn = targetArn,
                Id = targetID
            }
        };

        // Add the targets to the rule.
        var response = await _amazonEventBridge.PutTargetsAsync(
            new PutTargetsRequest()
            {
                EventBusName = eventBusArn,
                Rule = ruleName,
                Targets = targets,
            });

        if (response.FailedEntryCount > 0)
        {
            response.FailedEntries.ForEach(e =>
            {
                _logger.LogError(
                    $"Failed to add target {e.TargetId}: {e.ErrorMessage}, code
{e.ErrorCode}");
            });
        }

        return targetID;
    }
```

ルールのターゲットにインプットトランスフォーマーを追加します。

```
    /// <summary>
    /// Update an Amazon S3 object created rule with a transform on the target.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <param name="targetArn">The ARN of the target.</param>
    /// <param name="eventBusArn">Optional event bus ARN. If empty, uses the default
event bus.</param>
    /// <returns>The ID of the target.</returns>
    public async Task<string> UpdateS3UploadRuleTargetWithTransform(string ruleName,
string targetArn, string? eventBusArn = null)
```

```
    {
        var targetID = Guid.NewGuid().ToString();

        var targets = new List<Target>
        {
            new Target()
            {
                Id = targetID,
                Arn = targetArn,
                InputTransformer = new InputTransformer()
                {
                    InputPathsMap = new Dictionary<string, string>()
                    {
                        {"bucket", "$.detail.bucket.name"},
                        {"time", "$.time"}
                    },
                    InputTemplate = "\"Notification: an object was uploaded to
bucket <bucket> at <time>.\""
                }
            }
        };
        var response = await _amazonEventBridge.PutTargetsAsync(
            new PutTargetsRequest()
            {
                EventBusName = eventBusArn,
                Rule = ruleName,
                Targets = targets,
            });
        if (response.FailedEntryCount > 0)
        {
            response.FailedEntries.ForEach(e =>
            {
                _logger.LogError(
                    $"Failed to add target {e.TargetId}: {e.ErrorMessage}, code
{e.ErrorCode}");
            });
        }
        return targetID;
    }
```

- API の詳細については、「 API リファレンスPutTargets」の「」を参照してください。 AWS SDK for .NET

`RemoveTargets`

次の例は、RemoveTargets を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ルール名を使用してルールのすべてのターゲットを削除します。

```
    /// <summary>
    /// Delete an event rule by name.
    /// </summary>
    /// <param name="ruleName">The name of the event rule.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> RemoveAllTargetsFromRule(string ruleName)
    {
        var targetIds = new List<string>();
        var request = new ListTargetsByRuleRequest()
        {
            Rule = ruleName
        };
        ListTargetsByRuleResponse targetsResponse;
        do
        {
            targetsResponse = await
_amazonEventBridge.ListTargetsByRuleAsync(request);
            targetIds.AddRange(targetsResponse.Targets.Select(t => t.Id));
            request.NextToken = targetsResponse.NextToken;

        } while (targetsResponse.NextToken is not null);

        var removeResponse = await _amazonEventBridge.RemoveTargetsAsync(
            new RemoveTargetsRequest()
            {
                Rule = ruleName,
                Ids = targetIds
            });
```

```
        if (removeResponse.FailedEntryCount > 0)
        {
            removeResponse.FailedEntries.ForEach(e =>
            {
                _logger.LogError(
                    $"Failed to remove target {e.TargetId}: {e.ErrorMessage}, code
{e.ErrorCode}");
            });
        }

        return removeResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスRemoveTargets」の「」を参照してください。AWS SDK for .NET

シナリオ

ルールとターゲットの使用開始

次のコードサンプルは、以下の操作方法を示しています。

- ルールを作成して、ターゲットを追加する。
- ルールを有効化および無効化する。
- ルールとターゲットを一覧表示して更新する。
- イベントを送信して、リソースをクリーンアップする。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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のシナリオを実行します。

```
public class EventBridgeScenario
{
```

```
   /*
   Before running this .NET code example, set up your development environment,
including your credentials.

   This .NET example performs the following tasks with Amazon EventBridge:
   - Create a rule.
   - Add a target to a rule.
   - Enable and disable rules.
   - List rules and targets.
   - Update rules and targets.
   - Send events.
   - Delete the rule.
   */

   private static ILogger logger = null!;
   private static EventBridgeWrapper _eventBridgeWrapper = null!;
   private static IConfiguration _configuration = null!;

   private static IAmazonIdentityManagementService? _iamClient = null!;
   private static IAmazonSimpleNotificationService? _snsClient = null!;
   private static IAmazonS3 _s3Client = null!;

   static async Task Main(string[] args)
   {
       // Set up dependency injection for Amazon EventBridge.
       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<IAmazonEventBridge>()
           .AddAWSService<IAmazonIdentityManagementService>()
           .AddAWSService<IAmazonS3>()
           .AddAWSService<IAmazonSimpleNotificationService>()
           .AddTransient<EventBridgeWrapper>()
           )
           .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<EventBridgeScenario>();

        ServicesSetup(host);

        string topicArn = "";
        string roleArn = "";

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the Amazon EventBridge example scenario.");
        Console.WriteLine(new string('-', 80));

        try
        {
            roleArn = await CreateRole();

            await CreateBucketWithEventBridgeEvents();

            await AddEventRule(roleArn);

            await ListEventRules();

            topicArn = await CreateSnsTopic();

            var email = await SubscribeToSnsTopic(topicArn);

            await AddSnsTarget(topicArn);

            await ListTargets();

            await ListRulesForTarget(topicArn);

            await UploadS3File(_s3Client);

            await ChangeRuleState(false);

            await GetRuleState();

            await UpdateSnsEventRule(topicArn);

            await ChangeRuleState(true);
```

```
            await UploadS3File(_s3Client);

            await UpdateToCustomRule(topicArn);

            await TriggerCustomRule(email);

            await CleanupResources(topicArn);
        }
        catch (Exception ex)
        {
            logger.LogError(ex, "There was a problem executing the scenario.");
            await CleanupResources(topicArn);
        }
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("The Amazon EventBridge example scenario is complete.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Populate the services for use within the console application.
    /// </summary>
    /// <param name="host">The services host.</param>
    private static void ServicesSetup(IHost host)
    {
        _eventBridgeWrapper =
host.Services.GetRequiredService<EventBridgeWrapper>();
        _snsClient =
host.Services.GetRequiredService<IAmazonSimpleNotificationService>();
        _s3Client = host.Services.GetRequiredService<IAmazonS3>();
        _iamClient =
host.Services.GetRequiredService<IAmazonIdentityManagementService>();
    }

    /// <summary>
    /// Create a role to be used by EventBridge.
    /// </summary>
    /// <returns>The role Amazon Resource Name (ARN).</returns>
    public static async Task<string> CreateRole()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Creating a role to use with EventBridge and attaching
managed policy AmazonEventBridgeFullAccess.");
        Console.WriteLine(new string('-', 80));
```

```
        var roleName = _configuration["roleName"];

        var assumeRolePolicy = "{" +
                                "\"Version\": \"2012-10-17\"," +
                                "\"Statement\": [{" +
                                "\"Effect\": \"Allow\"," +
                                "\"Principal\": {" +
                                $"\"Service\": \"events.amazonaws.com\"" +
                                "}," +
                                "\"Action\": \"sts:AssumeRole\"" +
                                "}]" +
                                "}";

        var roleResult = await _iamClient!.CreateRoleAsync(
            new CreateRoleRequest()
            {
                AssumeRolePolicyDocument = assumeRolePolicy,
                Path = "/",
                RoleName = roleName
            });

        await _iamClient.AttachRolePolicyAsync(
            new AttachRolePolicyRequest()
            {
                PolicyArn = "arn:aws:iam::aws:policy/AmazonEventBridgeFullAccess",
                RoleName = roleName
            });
        // Allow time for the role to be ready.
        Thread.Sleep(10000);
        return roleResult.Role.Arn;
    }

    /// <summary>
    /// Create an Amazon Simple Storage Service (Amazon S3) bucket with EventBridge
events enabled.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CreateBucketWithEventBridgeEvents()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Creating an S3 bucket with EventBridge events enabled.");

        var testBucketName = _configuration["testBucketName"];
```

```
        var bucketExists = await
Amazon.S3.Util.AmazonS3Util.DoesS3BucketExistV2Async(_s3Client,
            testBucketName);

        if (!bucketExists)
        {
            await _s3Client.PutBucketAsync(new PutBucketRequest()
            {
                BucketName = testBucketName,
                UseClientRegion = true
            });
        }

        await _s3Client.PutBucketNotificationAsync(new
PutBucketNotificationRequest()
        {
            BucketName = testBucketName,
            EventBridgeConfiguration = new EventBridgeConfiguration()
        });

        Console.WriteLine($"\tAdded bucket {testBucketName} with EventBridge events
enabled.");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Create and upload a file to an S3 bucket to trigger an event.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task UploadS3File(IAmazonS3 s3Client)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Uploading a file to the test bucket. This will trigger a
subscription email.");

        var testBucketName = _configuration["testBucketName"];

        var fileName = $"example_upload_{DateTime.UtcNow.Ticks}.txt";

        // Create the file if it does not already exist.
        if (!File.Exists(fileName))
        {
```

```
            await using StreamWriter sw = File.CreateText(fileName);
            await sw.WriteLineAsync(
                "This is a sample file for testing uploads.");
        }

        await s3Client.PutObjectAsync(new PutObjectRequest()
        {
            FilePath = fileName,
            BucketName = testBucketName
        });

        Console.WriteLine($"\tPress Enter to continue.");
        Console.ReadLine();

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Create an Amazon Simple Notification Service (Amazon SNS) topic to use as an
EventBridge target.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task<string> CreateSnsTopic()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine(
            "Creating an Amazon Simple Notification Service (Amazon SNS) topic for
email subscriptions.");

        var topicName = _configuration["topicName"];

        string topicPolicy = "{" +
                             "\"Version\": \"2012-10-17\"," +
                             "\"Statement\": [{" +
                             "\"Sid\": \"EventBridgePublishTopic\"," +
                             "\"Effect\": \"Allow\"," +
                             "\"Principal\": {" +
                             $"\"Service\": \"events.amazonaws.com\"" +
                             "}," +
                             "\"Resource\": \"*\"," +
                             "\"Action\": \"sns:Publish\"" +
                             "}]" +
                             "}";
```

```
        var topicAttributes = new Dictionary<string, string>()
        {
            { "Policy", topicPolicy }
        };

        var topicResponse = await _snsClient!.CreateTopicAsync(new
CreateTopicRequest()
        {
            Name = topicName,
            Attributes = topicAttributes

        });

        Console.WriteLine($"\tAdded topic {topicName} for email subscriptions.");

        Console.WriteLine(new string('-', 80));

        return topicResponse.TopicArn;
    }

    /// <summary>
    /// Subscribe a user email to an SNS topic.
    /// </summary>
    /// <param name="topicArn">The ARN of the SNS topic.</param>
    /// <returns>The user's email.</returns>
    private static async Task<string> SubscribeToSnsTopic(string topicArn)
    {
        Console.WriteLine(new string('-', 80));


        string email = "";
        while (string.IsNullOrEmpty(email))
        {
            Console.WriteLine("Enter your email to subscribe to the Amazon SNS
topic:");
            email = Console.ReadLine()!;
        }

        var subscriptions = new List<string>();
        var paginatedSubscriptions =
_snsClient!.Paginators.ListSubscriptionsByTopic(
            new ListSubscriptionsByTopicRequest()
            {
                TopicArn = topicArn
```

```
            });

        // Get the entire list using the paginator.
        await foreach (var subscription in paginatedSubscriptions.Subscriptions)
        {
            subscriptions.Add(subscription.Endpoint);
        }

        if (subscriptions.Contains(email))
        {
            Console.WriteLine($"\tYour email is already subscribed.");
            Console.WriteLine(new string('-', 80));
            return email;
        }

        await _snsClient.SubscribeAsync(new SubscribeRequest()
        {
            TopicArn = topicArn,
            Protocol = "email",
            Endpoint = email
        });

        Console.WriteLine($"Use the link in the email you received to confirm your
subscription, then press Enter to continue.");

        Console.ReadLine();

        Console.WriteLine(new string('-', 80));
        return email;
    }

    /// <summary>
    /// Add a rule which triggers when a file is uploaded to an S3 bucket.
    /// </summary>
    /// <param name="roleArn">The ARN of the role used by EventBridge.</param>
    /// <returns>Async task.</returns>
    private static async Task AddEventRule(string roleArn)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Creating an EventBridge event that sends an email when an
Amazon S3 object is created.");

        var eventRuleName = _configuration["eventRuleName"];
        var testBucketName = _configuration["testBucketName"];
```

```
        await _eventBridgeWrapper.PutS3UploadRule(roleArn, eventRuleName,
testBucketName);
        Console.WriteLine($"\tAdded event rule {eventRuleName} for bucket
{testBucketName}.");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Add an SNS target to the rule.
    /// </summary>
    /// <param name="topicArn">The ARN of the SNS topic.</param>
    /// <returns>Async task.</returns>
    private static async Task AddSnsTarget(string topicArn)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Adding a target to the rule to that sends an email when
the rule is triggered.");

        var eventRuleName = _configuration["eventRuleName"];
        var testBucketName = _configuration["testBucketName"];
        var topicName = _configuration["topicName"];
        await _eventBridgeWrapper.AddSnsTargetToRule(eventRuleName, topicArn);
        Console.WriteLine($"\tAdded event rule {eventRuleName} with Amazon SNS
target {topicName} for bucket {testBucketName}.");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List the event rules on the default event bus.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task ListEventRules()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Current event rules:");

        var rules = await _eventBridgeWrapper.ListAllRulesForEventBus();
        rules.ForEach(r => Console.WriteLine($"\tRule: {r.Name} Description:
{r.Description} State: {r.State}"));

        Console.WriteLine(new string('-', 80));
```

```
    }

    /// <summary>
    /// Update the event target to use a transform.
    /// </summary>
    /// <param name="topicArn">The SNS topic ARN target to update.</param>
    /// <returns>Async task.</returns>
    private static async Task UpdateSnsEventRule(string topicArn)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Let's update the event target with a transform.");

        var eventRuleName = _configuration["eventRuleName"];
        var testBucketName = _configuration["testBucketName"];

        await
_eventBridgeWrapper.UpdateS3UploadRuleTargetWithTransform(eventRuleName, topicArn);
        Console.WriteLine($"\tUpdated event rule {eventRuleName} with Amazon SNS
target {topicArn} for bucket {testBucketName}.");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Update the rule to use a custom event pattern.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task UpdateToCustomRule(string topicArn)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Updating the event pattern to be triggered by a custom
event instead.");

        var eventRuleName = _configuration["eventRuleName"];

        await _eventBridgeWrapper.UpdateCustomEventPattern(eventRuleName);

        Console.WriteLine($"\tUpdated event rule {eventRuleName} to custom
pattern.");
        await _eventBridgeWrapper.UpdateCustomRuleTargetWithTransform(eventRuleName,
            topicArn);

        Console.WriteLine($"\tUpdated event target {topicArn}.");
```

```
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Send rule events for a custom rule using the user's email address.
    /// </summary>
    /// <param name="email">The email address to include.</param>
    /// <returns>Async task.</returns>
    private static async Task TriggerCustomRule(string email)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Sending an event to trigger the rule. This will trigger a
subscription email.");

        await _eventBridgeWrapper.PutCustomEmailEvent(email);

        Console.WriteLine($"\tEvents have been sent. Press Enter to continue.");
        Console.ReadLine();

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List all of the targets for a rule.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task ListTargets()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("List all of the targets for a particular rule.");

        var eventRuleName = _configuration["eventRuleName"];
        var targets = await _eventBridgeWrapper.ListAllTargetsOnRule(eventRuleName);
        targets.ForEach(t => Console.WriteLine($"\tTarget: {t.Arn} Id: {t.Id} Input:
{t.Input}"));

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List all of the rules for a particular target.
    /// </summary>
    /// <param name="topicArn">The ARN of the SNS topic.</param>
    /// <returns>Async task.</returns>
```

```
    private static async Task ListRulesForTarget(string topicArn)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("List all of the rules for a particular target.");

        var rules = await _eventBridgeWrapper.ListAllRuleNamesByTarget(topicArn);
        rules.ForEach(r => Console.WriteLine($"\tRule: {r}"));

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Enable or disable a particular rule.
    /// </summary>
    /// <param name="isEnabled">True to enable the rule, otherwise false.</param>
    /// <returns>Async task.</returns>
    private static async Task ChangeRuleState(bool isEnabled)
    {
        Console.WriteLine(new string('-', 80));
        var eventRuleName = _configuration["eventRuleName"];

        if (!isEnabled)
        {
            Console.WriteLine($"Disabling the rule: {eventRuleName}");
            await _eventBridgeWrapper.DisableRuleByName(eventRuleName);
        }
        else
        {
            Console.WriteLine($"Enabling the rule: {eventRuleName}");
            await _eventBridgeWrapper.EnableRuleByName(eventRuleName);
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get the current state of the rule.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetRuleState()
    {
        Console.WriteLine(new string('-', 80));
        var eventRuleName = _configuration["eventRuleName"];
```

```
        var state = await _eventBridgeWrapper.GetRuleStateByRuleName(eventRuleName);
        Console.WriteLine($"Rule {eventRuleName} is in current state {state}.");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <param name="topicArn">The ARN of the SNS topic to clean up.</param>
    /// <returns>Async task.</returns>
    private static async Task CleanupResources(string topicArn)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Clean up resources.");

        var eventRuleName = _configuration["eventRuleName"];
        if (GetYesNoResponse($"\tDelete all targets and event rule {eventRuleName}?
(y/n)"))
        {
            Console.WriteLine($"\tRemoving all targets from the event rule.");
            await _eventBridgeWrapper.RemoveAllTargetsFromRule(eventRuleName);

            Console.WriteLine($"\tDeleting event rule.");
            await _eventBridgeWrapper.DeleteRuleByName(eventRuleName);
        }

        var topicName = _configuration["topicName"];
        if (GetYesNoResponse($"\tDelete Amazon SNS subscription topic {topicName}?
(y/n)"))
        {
            Console.WriteLine($"\tDeleting topic.");
            await _snsClient!.DeleteTopicAsync(new DeleteTopicRequest()
            {
                TopicArn = topicArn
            });
        }

        var bucketName = _configuration["testBucketName"];
        if (GetYesNoResponse($"\tDelete Amazon S3 bucket {bucketName}? (y/n)"))
        {
            Console.WriteLine($"\tDeleting bucket.");
            // Delete all objects in the bucket.
```

```
            var deleteList = await _s3Client.ListObjectsV2Async(new
ListObjectsV2Request()
            {
                BucketName = bucketName
            });
            await _s3Client.DeleteObjectsAsync(new DeleteObjectsRequest()
            {
                BucketName = bucketName,
                Objects = deleteList.S3Objects
                    .Select(o => new KeyVersion { Key = o.Key }).ToList()
            });
            // Now delete the bucket.
            await _s3Client.DeleteBucketAsync(new DeleteBucketRequest()
            {
                BucketName = bucketName
            });
        }

        var roleName = _configuration["roleName"];
        if (GetYesNoResponse($"\tDelete role {roleName}? (y/n)"))
        {
            Console.WriteLine($"\tDetaching policy and deleting role.");

            await _iamClient!.DetachRolePolicyAsync(new DetachRolePolicyRequest()
            {
                RoleName = roleName,
                PolicyArn = "arn:aws:iam::aws:policy/AmazonEventBridgeFullAccess",
            });

            await _iamClient!.DeleteRoleAsync(new DeleteRoleRequest()
            {
                RoleName = roleName
            });
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Helper method to 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;
    }
}
```

EventBridge オペレーションをラップするクラスを作成します。

```
/// <summary>
/// Wrapper for Amazon EventBridge operations.
/// </summary>
public class EventBridgeWrapper
{
    private readonly IAmazonEventBridge _amazonEventBridge;
    private readonly ILogger<EventBridgeWrapper> _logger;

    /// <summary>
    /// Constructor for the EventBridge wrapper.
    /// </summary>
    /// <param name="amazonEventBridge">The injected EventBridge client.</param>
    /// <param name="logger">The injected logger for the wrapper.</param>
    public EventBridgeWrapper(IAmazonEventBridge amazonEventBridge,
 ILogger<EventBridgeWrapper> logger)

    {
        _amazonEventBridge = amazonEventBridge;
        _logger = logger;
    }

    /// <summary>
    /// Get the state for a rule by the rule name.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <param name="eventBusName">The optional name of the event bus. If empty,
 uses the default event bus.</param>
    /// <returns>The state of the rule.</returns>
```

```
    public async Task<RuleState> GetRuleStateByRuleName(string ruleName, string?
eventBusName = null)
    {
        var ruleResponse = await _amazonEventBridge.DescribeRuleAsync(
            new DescribeRuleRequest()
            {
                Name = ruleName,
                EventBusName = eventBusName
            });
        return ruleResponse.State;
    }

    /// <summary>
    /// Enable a particular rule on an event bus.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> EnableRuleByName(string ruleName)
    {
        var ruleResponse = await _amazonEventBridge.EnableRuleAsync(
            new EnableRuleRequest()
            {
                Name = ruleName
            });
        return ruleResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Disable a particular rule on an event bus.
    /// </summary
    /// <param name="ruleName">The name of the rule.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DisableRuleByName(string ruleName)
    {
        var ruleResponse = await _amazonEventBridge.DisableRuleAsync(
            new DisableRuleRequest()
            {
                Name = ruleName
            });
        return ruleResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// List the rules on an event bus.
```

```
    /// </summary>
    /// <param name="eventBusArn">The optional ARN of the event bus. If empty, uses
the default event bus.</param>
    /// <returns>The list of rules.</returns>
    public async Task<List<Rule>> ListAllRulesForEventBus(string? eventBusArn =
null)
    {
        var results = new List<Rule>();
        var request = new ListRulesRequest()
        {
            EventBusName = eventBusArn
        };
        // Get all of the pages of rules.
        ListRulesResponse response;
        do
        {
            response = await _amazonEventBridge.ListRulesAsync(request);
            results.AddRange(response.Rules);
            request.NextToken = response.NextToken;

        } while (response.NextToken is not null);

        return results;
    }

    /// <summary>
    /// List all of the targets matching a rule by name.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <returns>The list of targets.</returns>
    public async Task<List<Target>> ListAllTargetsOnRule(string ruleName)
    {
        var results = new List<Target>();
        var request = new ListTargetsByRuleRequest()
        {
            Rule = ruleName
        };
        ListTargetsByRuleResponse response;
        do
        {
            response = await _amazonEventBridge.ListTargetsByRuleAsync(request);
            results.AddRange(response.Targets);
            request.NextToken = response.NextToken;
```

```
        } while (response.NextToken is not null);

        return results;
    }

    /// <summary>
    /// List names of all rules matching a target.
    /// </summary>
    /// <param name="targetArn">The ARN of the target.</param>
    /// <returns>The list of rule names.</returns>
    public async Task<List<string>> ListAllRuleNamesByTarget(string targetArn)
    {
        var results = new List<string>();
        var request = new ListRuleNamesByTargetRequest()
        {
            TargetArn = targetArn
        };
        ListRuleNamesByTargetResponse response;
        do
        {
            response = await _amazonEventBridge.ListRuleNamesByTargetAsync(request);
            results.AddRange(response.RuleNames);
            request.NextToken = response.NextToken;

        } while (response.NextToken is not null);

        return results;
    }

    /// <summary>
    /// Create a new event rule that triggers when an Amazon S3 object is created in
 a bucket.
    /// </summary>
    /// <param name="roleArn">The ARN of the role.</param>
    /// <param name="ruleName">The name to give the rule.</param>
    /// <param name="bucketName">The name of the bucket to trigger the event.</
param>
    /// <returns>The ARN of the new rule.</returns>
    public async Task<string> PutS3UploadRule(string roleArn, string ruleName,
 string bucketName)
    {
        string eventPattern = "{" +
                              "\"source\": [\"aws.s3\"]," +
                                  "\"detail-type\": [\"Object Created\"]," +
```

```
                                    "\"detail\": {" +
                                        "\"bucket\": {" +
                                            "\"name\": [\"" + bucketName + "\"]" +
                                        "}" +
                                    "}" +
                                "}";

        var response = await _amazonEventBridge.PutRuleAsync(
            new PutRuleRequest()
            {
                Name = ruleName,
                Description = "Example S3 upload rule for EventBridge",
                RoleArn = roleArn,
                EventPattern = eventPattern
            });

        return response.RuleArn;
    }

    /// <summary>
    /// Update an Amazon S3 object created rule with a transform on the target.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <param name="targetArn">The ARN of the target.</param>
    /// <param name="eventBusArn">Optional event bus ARN. If empty, uses the default
event bus.</param>
    /// <returns>The ID of the target.</returns>
    public async Task<string> UpdateS3UploadRuleTargetWithTransform(string ruleName,
string targetArn, string? eventBusArn = null)
    {
        var targetID = Guid.NewGuid().ToString();

        var targets = new List<Target>
        {
            new Target()
            {
                Id = targetID,
                Arn = targetArn,
                InputTransformer = new InputTransformer()
                {
                    InputPathsMap = new Dictionary<string, string>()
                    {
                        {"bucket", "$.detail.bucket.name"},
                        {"time", "$.time"}
```

```
                    },
                    InputTemplate = "\"Notification: an object was uploaded to
bucket <bucket> at <time>.\""
                }
            }
        };
        var response = await _amazonEventBridge.PutTargetsAsync(
            new PutTargetsRequest()
            {
                EventBusName = eventBusArn,
                Rule = ruleName,
                Targets = targets,
            });
        if (response.FailedEntryCount > 0)
        {
            response.FailedEntries.ForEach(e =>
            {
                _logger.LogError(
                    $"Failed to add target {e.TargetId}: {e.ErrorMessage}, code
{e.ErrorCode}");
            });
        }
        return targetID;
    }

    /// <summary>
    /// Update a custom rule with a transform on the target.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <param name="targetArn">The ARN of the target.</param>
    /// <param name="eventBusArn">Optional event bus ARN. If empty, uses the default
event bus.</param>
    /// <returns>The ID of the target.</returns>
    public async Task<string> UpdateCustomRuleTargetWithTransform(string ruleName,
string targetArn, string? eventBusArn = null)
    {
        var targetID = Guid.NewGuid().ToString();

        var targets = new List<Target>
        {
            new Target()
            {
                Id = targetID,
                Arn = targetArn,
```

```
                InputTransformer = new InputTransformer()
                {
                    InputTemplate = "\"Notification: sample event was received.\""
                }
            }
        };
        var response = await _amazonEventBridge.PutTargetsAsync(
            new PutTargetsRequest()
            {
                EventBusName = eventBusArn,
                Rule = ruleName,
                Targets = targets,
            });
        if (response.FailedEntryCount > 0)
        {
            response.FailedEntries.ForEach(e =>
            {
                _logger.LogError(
                    $"Failed to add target {e.TargetId}: {e.ErrorMessage}, code
{e.ErrorCode}");
            });
        }
        return targetID;
    }

    /// <summary>
    /// Add an event to the event bus that includes an email, message, and time.
    /// </summary>
    /// <param name="email">The email to use in the event detail of the custom
event.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutCustomEmailEvent(string email)
    {
        var eventDetail = new
        {
            UserEmail = email,
            Message = "This event was generated by example code.",
            UtcTime = DateTime.UtcNow.ToString("g")
        };
        var response = await _amazonEventBridge.PutEventsAsync(
            new PutEventsRequest()
            {
                Entries = new List<PutEventsRequestEntry>()
                {
```

```
                    new PutEventsRequestEntry()
                    {
                        Source = "ExampleSource",
                        Detail = JsonSerializer.Serialize(eventDetail),
                        DetailType = "ExampleType"
                    }
                }
            });

        return response.FailedEntryCount == 0;
    }

    /// <summary>
    /// Update a rule to use a custom defined event pattern.
    /// </summary>
    /// <param name="ruleName">The name of the rule to update.</param>
    /// <returns>The ARN of the updated rule.</returns>
    public async Task<string> UpdateCustomEventPattern(string ruleName)
    {
        string customEventsPattern = "{" +
                                     "\"source\": [\"ExampleSource\"]," +
                                     "\"detail-type\": [\"ExampleType\"]" +
                                     "}";

        var response = await _amazonEventBridge.PutRuleAsync(
            new PutRuleRequest()
            {
                Name = ruleName,
                Description = "Custom test rule",
                EventPattern = customEventsPattern
            });

        return response.RuleArn;
    }

    /// <summary>
    /// Add an Amazon SNS target topic to a rule.
    /// </summary>
    /// <param name="ruleName">The name of the rule to update.</param>
    /// <param name="targetArn">The ARN of the Amazon SNS target.</param>
    /// <param name="eventBusArn">The optional event bus name, uses default if
empty.</param>
    /// <returns>The ID of the target.</returns>
```

```
    public async Task<string> AddSnsTargetToRule(string ruleName, string targetArn,
string? eventBusArn = null)
    {
        var targetID = Guid.NewGuid().ToString();

        // Create the list of targets and add a new target.
        var targets = new List<Target>
        {
            new Target()
            {
                Arn = targetArn,
                Id = targetID
            }
        };

        // Add the targets to the rule.
        var response = await _amazonEventBridge.PutTargetsAsync(
            new PutTargetsRequest()
            {
                EventBusName = eventBusArn,
                Rule = ruleName,
                Targets = targets,
            });

        if (response.FailedEntryCount > 0)
        {
            response.FailedEntries.ForEach(e =>
            {
                _logger.LogError(
                    $"Failed to add target {e.TargetId}: {e.ErrorMessage}, code
{e.ErrorCode}");
            });
        }

        return targetID;
    }

    /// <summary>
    /// Delete an event rule by name.
    /// </summary>
    /// <param name="ruleName">The name of the event rule.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> RemoveAllTargetsFromRule(string ruleName)
    {
```

```
        var targetIds = new List<string>();
        var request = new ListTargetsByRuleRequest()
        {
            Rule = ruleName
        };
        ListTargetsByRuleResponse targetsResponse;
        do
        {
            targetsResponse = await
_amazonEventBridge.ListTargetsByRuleAsync(request);
            targetIds.AddRange(targetsResponse.Targets.Select(t => t.Id));
            request.NextToken = targetsResponse.NextToken;

        } while (targetsResponse.NextToken is not null);

        var removeResponse = await _amazonEventBridge.RemoveTargetsAsync(
            new RemoveTargetsRequest()
            {
                Rule = ruleName,
                Ids = targetIds
            });

        if (removeResponse.FailedEntryCount > 0)
        {
            removeResponse.FailedEntries.ForEach(e =>
            {
                _logger.LogError(
                    $"Failed to remove target {e.TargetId}: {e.ErrorMessage}, code
{e.ErrorCode}");
            });
        }

        return removeResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete an event rule by name.
    /// </summary>
    /// <param name="ruleName">The name of the event rule.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteRuleByName(string ruleName)
    {
        var response = await _amazonEventBridge.DeleteRuleAsync(
            new DeleteRuleRequest()
```

```
            {
                Name = ruleName
            });

        return response.HttpStatusCode == HttpStatusCode.OK;
    }
}
```

- 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。
  - DeleteRule
  - DescribeRule
  - DisableRule
  - EnableRule
  - ListRuleNamesByTarget
  - ListRules
  - ListTargetsByRule
  - PutEvents
  - PutRule
  - PutTargets

AWS Glue を使用した の例 AWS SDK for .NET

次のコード例は、 AWS SDK for .NET で 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 Glue。

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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。このリンクには GitHub、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が記載されています。

開始方法

こんにち AWS Glueは

次のコード例は、 AWS Glueの使用を開始する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
namespace GlueActions;

public class HelloGlue
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for AWS Glue.
        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<IAmazonGlue>()
                .AddTransient<GlueWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<HelloGlue>();
        var glueClient = host.Services.GetRequiredService<IAmazonGlue>();

        var request = new ListJobsRequest();

        var jobNames = new List<string>();

        do
        {
```

```
            var response = await glueClient.ListJobsAsync(request);
            jobNames.AddRange(response.JobNames);
            request.NextToken = response.NextToken;
        }
        while (request.NextToken is not null);

        Console.Clear();
        Console.WriteLine("Hello, Glue. Let's list your existing Glue Jobs:");
        if (jobNames.Count == 0)
        {
            Console.WriteLine("You don't have any AWS Glue jobs.");
        }
        else
        {
            jobNames.ForEach(Console.WriteLine);
        }
    }
}
```

- API の詳細については、「 API リファレンスListJobs」の「」を参照してください。 AWS SDK for .NET

トピック

- アクション
- シナリオ

アクション

`CreateCrawler`

次の例は、`CreateCrawler` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Create an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name for the crawler.</param>
    /// <param name="crawlerDescription">A description of the crawler.</param>
    /// <param name="role">The AWS Identity and Access Management (IAM) role to
    /// be assumed by the crawler.</param>
    /// <param name="schedule">The schedule on which the crawler will be executed.</
param>
    /// <param name="s3Path">The path to the Amazon Simple Storage Service (Amazon
 S3)
    /// bucket where the Python script has been stored.</param>
    /// <param name="dbName">The name to use for the database that will be
    /// created by the crawler.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> CreateCrawlerAsync(
        string crawlerName,
        string crawlerDescription,
        string role,
        string schedule,
        string s3Path,
        string dbName)
    {
        var s3Target = new S3Target
        {
            Path = s3Path,
        };

        var targetList = new List<S3Target>
        {
            s3Target,
        };

        var targets = new CrawlerTargets
        {
            S3Targets = targetList,
        };

        var crawlerRequest = new CreateCrawlerRequest
        {
            DatabaseName = dbName,
            Name = crawlerName,
            Description = crawlerDescription,
```

```
            Targets = targets,
            Role = role,
            Schedule = schedule,
        };

        var response = await _amazonGlue.CreateCrawlerAsync(crawlerRequest);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスCreateCrawler」の「」を参照してください。AWS SDK for .NET

`CreateJob`

次の例は、CreateJob を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Create an AWS Glue job.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <param name="roleName">The name of the IAM role to be assumed by
    /// the job.</param>
    /// <param name="description">A description of the job.</param>
    /// <param name="scriptUrl">The URL to the script.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> CreateJobAsync(string dbName, string tableName, string
bucketUrl, string jobName, string roleName, string description, string scriptUrl)
    {
        var command = new JobCommand
        {
            PythonVersion = "3",
```

```
            Name = "glueetl",
            ScriptLocation = scriptUrl,
        };

        var arguments = new Dictionary<string, string>
        {
            { "--input_database", dbName },
            { "--input_table", tableName },
            { "--output_bucket_url", bucketUrl }
        };

        var request = new CreateJobRequest
        {
            Command = command,
            DefaultArguments = arguments,
            Description = description,
            GlueVersion = "3.0",
            Name = jobName,
            NumberOfWorkers = 10,
            Role = roleName,
            WorkerType = "G.1X"
        };

        var response = await _amazonGlue.CreateJobAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスCreateJob」の「」を参照してください。 AWS SDK for .NET

`DeleteCrawler`

次の例は、`DeleteCrawler` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name of the crawler.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteCrawlerAsync(string crawlerName)
    {
        var response = await _amazonGlue.DeleteCrawlerAsync(new DeleteCrawlerRequest
{ Name = crawlerName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスDeleteCrawler」の「」を参照してください。AWS SDK for .NET

`DeleteDatabase`

次の例は、`DeleteDatabase` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete the AWS Glue database.
```

```
    /// </summary>
    /// <param name="dbName">The name of the database.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteDatabaseAsync(string dbName)
    {
        var response = await _amazonGlue.DeleteDatabaseAsync(new
DeleteDatabaseRequest { Name = dbName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスDeleteDatabase」の「」を参照してください。AWS SDK for .NET

`DeleteJob`

次の例は、DeleteJob を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete an AWS Glue job.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteJobAsync(string jobName)
    {
        var response = await _amazonGlue.DeleteJobAsync(new DeleteJobRequest
{ JobName = jobName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスDeleteJob」の「」を参照してください。 AWS SDK for .NET

`DeleteTable`

次の例は、`DeleteTable` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete a table from an AWS Glue database.
    /// </summary>
    /// <param name="tableName">The table to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteTableAsync(string dbName, string tableName)
    {
        var response = await _amazonGlue.DeleteTableAsync(new DeleteTableRequest
{ Name = tableName, DatabaseName = dbName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスDeleteTable」の「」を参照してください。 AWS SDK for .NET

`GetCrawler`

次の例は、`GetCrawler` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get information about an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name of the crawler.</param>
    /// <returns>A Crawler object describing the crawler.</returns>
    public async Task<Crawler?> GetCrawlerAsync(string crawlerName)
    {
        var crawlerRequest = new GetCrawlerRequest
        {
            Name = crawlerName,
        };

        var response = await _amazonGlue.GetCrawlerAsync(crawlerRequest);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            var databaseName = response.Crawler.DatabaseName;
            Console.WriteLine($"{crawlerName} has the database {databaseName}");
            return response.Crawler;
        }

        Console.WriteLine($"No information regarding {crawlerName} could be
found.");
        return null;
    }
```

- API の詳細については、「 API リファレンスGetCrawler」の「」を参照してください。 AWS SDK for .NET

`GetDatabase`

次の例は、GetDatabase を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get information about an AWS Glue database.
    /// </summary>
    /// <param name="dbName">The name of the database.</param>
    /// <returns>A Database object containing information about the database.</
returns>
    public async Task<Database> GetDatabaseAsync(string dbName)
    {
        var databasesRequest = new GetDatabaseRequest
        {
            Name = dbName,
        };

        var response = await _amazonGlue.GetDatabaseAsync(databasesRequest);
        return response.Database;
    }
```

- API の詳細については、「 API リファレンスGetDatabase」の「」を参照してください。 AWS SDK for .NET

`GetJobRun`

次の例は、`GetJobRun` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get information about a specific AWS Glue job run.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <param name="jobRunId">The Id of the job run.</param>
    /// <returns>A JobRun object with information about the job run.</returns>
    public async Task<JobRun> GetJobRunAsync(string jobName, string jobRunId)
    {
        var response = await _amazonGlue.GetJobRunAsync(new GetJobRunRequest
{ JobName = jobName, RunId = jobRunId });
        return response.JobRun;
    }
```

- API の詳細については、「 API リファレンスGetJobRun」の「」を参照してください。 AWS SDK for .NET

`GetJobRuns`

次の例は、GetJobRuns を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get information about all AWS Glue runs of a specific job.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <returns>A list of JobRun objects.</returns>
    public async Task<List<JobRun>> GetJobRunsAsync(string jobName)
    {
        var jobRuns = new List<JobRun>();

        var request = new GetJobRunsRequest
        {
```

```
            JobName = jobName,
        };

        // No need to loop to get all the log groups--the SDK does it for us behind
the scenes
        var paginatorForJobRuns =
            _amazonGlue.Paginators.GetJobRuns(request);

        await foreach (var response in paginatorForJobRuns.Responses)
        {
            response.JobRuns.ForEach(jobRun =>
            {
                jobRuns.Add(jobRun);
            });
        }

        return jobRuns;
    }
```

- API の詳細については、「 API リファレンスGetJobRuns」の「」を参照してください。 AWS SDK for .NET

`GetTables`

次の例は、`GetTables` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get a list of tables for an AWS Glue database.
    /// </summary>
    /// <param name="dbName">The name of the database.</param>
    /// <returns>A list of Table objects.</returns>
```

```
    public async Task<List<Table>> GetTablesAsync(string dbName)
    {
        var request = new GetTablesRequest { DatabaseName = dbName };
        var tables = new List<Table>();

        // Get a paginator for listing the tables.
        var tablePaginator = _amazonGlue.Paginators.GetTables(request);

        await foreach (var response in tablePaginator.Responses)
        {
            tables.AddRange(response.TableList);
        }

        return tables;
    }
```

- API の詳細については、「 API リファレンスGetTables」の「」を参照してください。 AWS SDK for .NET

`ListJobs`

次の例は、`ListJobs` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// List AWS Glue jobs using a paginator.
    /// </summary>
    /// <returns>A list of AWS Glue job names.</returns>
    public async Task<List<string>> ListJobsAsync()
    {
        var jobNames = new List<string>();
```

```
        var listJobsPaginator = _amazonGlue.Paginators.ListJobs(new ListJobsRequest
{ MaxResults = 10 });
        await foreach (var response in listJobsPaginator.Responses)
        {
            jobNames.AddRange(response.JobNames);
        }

        return jobNames;
    }
```

- API の詳細については、「 API リファレンスListJobs」の「」を参照してください。 AWS SDK for .NET

`StartCrawler`

次の例は、`StartCrawler` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Start an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name of the crawler.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> StartCrawlerAsync(string crawlerName)
    {
        var crawlerRequest = new StartCrawlerRequest
        {
            Name = crawlerName,
        };

        var response = await _amazonGlue.StartCrawlerAsync(crawlerRequest);
```

```
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスStartCrawler」の「」を参照してください。 AWS SDK for .NET

`StartJobRun`

次の例は、StartJobRun を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Start an AWS Glue job run.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <returns>A string representing the job run Id.</returns>
    public async Task<string> StartJobRunAsync(
        string jobName,
        string inputDatabase,
        string inputTable,
        string bucketName)
    {
        var request = new StartJobRunRequest
        {
            JobName = jobName,
            Arguments = new Dictionary<string, string>
            {
                {"--input_database", inputDatabase},
                {"--input_table", inputTable},
                {"--output_bucket_url", $"s3://{bucketName}/"}
            }
        };
```

```
        var response = await _amazonGlue.StartJobRunAsync(request);
        return response.JobRunId;
    }
```

- API の詳細については、「 API リファレンスStartJobRun」の「」を参照してください。 AWS SDK for .NET

シナリオ

クローラーとジョブを開始する

次のコードサンプルは、以下の操作方法を示しています。

- パブリック Amazon S3 バケットをクロールし、CSV 形式のメタデータのデータベースを生成するクローラーを作成する。
- のデータベースとテーブルに関する情報を一覧表示します AWS Glue Data Catalog。
- S3 バケットから CSV 形式のデータを抽出するジョブを作成し、そのデータを変換して JSON 形式の出力を別の S3 バケットにロードする。
- ジョブ実行に関する情報を一覧表示し、変換されたデータを表示してリソースをクリーンアップする。

詳細については、「チュートリアル: AWS Glue Studio の開始方法」を参照してください。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

シナリオで使用される AWS Glue 関数をラップするクラスを作成します。

```
using System.Net;
```

```
namespace GlueActions;

public class GlueWrapper
{
    private readonly IAmazonGlue _amazonGlue;

    /// <summary>
    /// Constructor for the AWS Glue actions wrapper.
    /// </summary>
    /// <param name="amazonGlue"></param>
    public GlueWrapper(IAmazonGlue amazonGlue)
    {
        _amazonGlue = amazonGlue;
    }

    /// <summary>
    /// Create an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name for the crawler.</param>
    /// <param name="crawlerDescription">A description of the crawler.</param>
    /// <param name="role">The AWS Identity and Access Management (IAM) role to
    /// be assumed by the crawler.</param>
    /// <param name="schedule">The schedule on which the crawler will be executed.</
param>
    /// <param name="s3Path">The path to the Amazon Simple Storage Service (Amazon
 S3)
    /// bucket where the Python script has been stored.</param>
    /// <param name="dbName">The name to use for the database that will be
    /// created by the crawler.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> CreateCrawlerAsync(
        string crawlerName,
        string crawlerDescription,
        string role,
        string schedule,
        string s3Path,
        string dbName)
    {
        var s3Target = new S3Target
        {
            Path = s3Path,
        };

        var targetList = new List<S3Target>
```

```
        {
            s3Target,
        };

        var targets = new CrawlerTargets
        {
            S3Targets = targetList,
        };

        var crawlerRequest = new CreateCrawlerRequest
        {
            DatabaseName = dbName,
            Name = crawlerName,
            Description = crawlerDescription,
            Targets = targets,
            Role = role,
            Schedule = schedule,
        };

        var response = await _amazonGlue.CreateCrawlerAsync(crawlerRequest);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Create an AWS Glue job.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <param name="roleName">The name of the IAM role to be assumed by
    /// the job.</param>
    /// <param name="description">A description of the job.</param>
    /// <param name="scriptUrl">The URL to the script.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> CreateJobAsync(string dbName, string tableName, string
bucketUrl, string jobName, string roleName, string description, string scriptUrl)
    {
        var command = new JobCommand
        {
            PythonVersion = "3",
            Name = "glueetl",
            ScriptLocation = scriptUrl,
        };

        var arguments = new Dictionary<string, string>
```

```
        {
            { "--input_database", dbName },
            { "--input_table", tableName },
            { "--output_bucket_url", bucketUrl }
        };

        var request = new CreateJobRequest
        {
            Command = command,
            DefaultArguments = arguments,
            Description = description,
            GlueVersion = "3.0",
            Name = jobName,
            NumberOfWorkers = 10,
            Role = roleName,
            WorkerType = "G.1X"
        };

        var response = await _amazonGlue.CreateJobAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name of the crawler.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteCrawlerAsync(string crawlerName)
    {
        var response = await _amazonGlue.DeleteCrawlerAsync(new DeleteCrawlerRequest
{ Name = crawlerName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete the AWS Glue database.
    /// </summary>
    /// <param name="dbName">The name of the database.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteDatabaseAsync(string dbName)
    {
```

```
        var response = await _amazonGlue.DeleteDatabaseAsync(new
DeleteDatabaseRequest { Name = dbName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an AWS Glue job.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteJobAsync(string jobName)
    {
        var response = await _amazonGlue.DeleteJobAsync(new DeleteJobRequest
{ JobName = jobName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete a table from an AWS Glue database.
    /// </summary>
    /// <param name="tableName">The table to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteTableAsync(string dbName, string tableName)
    {
        var response = await _amazonGlue.DeleteTableAsync(new DeleteTableRequest
{ Name = tableName, DatabaseName = dbName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Get information about an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name of the crawler.</param>
    /// <returns>A Crawler object describing the crawler.</returns>
    public async Task<Crawler?> GetCrawlerAsync(string crawlerName)
    {
        var crawlerRequest = new GetCrawlerRequest
        {
            Name = crawlerName,
        };
```

```
        var response = await _amazonGlue.GetCrawlerAsync(crawlerRequest);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            var databaseName = response.Crawler.DatabaseName;
            Console.WriteLine($"{crawlerName} has the database {databaseName}");
            return response.Crawler;
        }

        Console.WriteLine($"No information regarding {crawlerName} could be
 found.");
        return null;
    }


    /// <summary>
    /// Get information about the state of an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name of the crawler.</param>
    /// <returns>A value describing the state of the crawler.</returns>
    public async Task<CrawlerState> GetCrawlerStateAsync(string crawlerName)
    {
        var response = await _amazonGlue.GetCrawlerAsync(
            new GetCrawlerRequest { Name = crawlerName });
        return response.Crawler.State;
    }


    /// <summary>
    /// Get information about an AWS Glue database.
    /// </summary>
    /// <param name="dbName">The name of the database.</param>
    /// <returns>A Database object containing information about the database.</
returns>
    public async Task<Database> GetDatabaseAsync(string dbName)
    {
        var databasesRequest = new GetDatabaseRequest
        {
            Name = dbName,
        };

        var response = await _amazonGlue.GetDatabaseAsync(databasesRequest);
        return response.Database;
    }
```

```
    /// <summary>
    /// Get information about a specific AWS Glue job run.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <param name="jobRunId">The Id of the job run.</param>
    /// <returns>A JobRun object with information about the job run.</returns>
    public async Task<JobRun> GetJobRunAsync(string jobName, string jobRunId)
    {
        var response = await _amazonGlue.GetJobRunAsync(new GetJobRunRequest
{ JobName = jobName, RunId = jobRunId });
        return response.JobRun;
    }


    /// <summary>
    /// Get information about all AWS Glue runs of a specific job.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <returns>A list of JobRun objects.</returns>
    public async Task<List<JobRun>> GetJobRunsAsync(string jobName)
    {
        var jobRuns = new List<JobRun>();

        var request = new GetJobRunsRequest
        {
            JobName = jobName,
        };

        // No need to loop to get all the log groups--the SDK does it for us behind
the scenes
        var paginatorForJobRuns =
            _amazonGlue.Paginators.GetJobRuns(request);

        await foreach (var response in paginatorForJobRuns.Responses)
        {
            response.JobRuns.ForEach(jobRun =>
            {
                jobRuns.Add(jobRun);
            });
        }

        return jobRuns;
    }
```

```
    /// <summary>
    /// Get a list of tables for an AWS Glue database.
    /// </summary>
    /// <param name="dbName">The name of the database.</param>
    /// <returns>A list of Table objects.</returns>
    public async Task<List<Table>> GetTablesAsync(string dbName)
    {
        var request = new GetTablesRequest { DatabaseName = dbName };
        var tables = new List<Table>();

        // Get a paginator for listing the tables.
        var tablePaginator = _amazonGlue.Paginators.GetTables(request);

        await foreach (var response in tablePaginator.Responses)
        {
            tables.AddRange(response.TableList);
        }

        return tables;
    }


    /// <summary>
    /// List AWS Glue jobs using a paginator.
    /// </summary>
    /// <returns>A list of AWS Glue job names.</returns>
    public async Task<List<string>> ListJobsAsync()
    {
        var jobNames = new List<string>();

        var listJobsPaginator = _amazonGlue.Paginators.ListJobs(new ListJobsRequest
{ MaxResults = 10 });
        await foreach (var response in listJobsPaginator.Responses)
        {
            jobNames.AddRange(response.JobNames);
        }

        return jobNames;
    }


    /// <summary>
```

```
    /// Start an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name of the crawler.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> StartCrawlerAsync(string crawlerName)
    {
        var crawlerRequest = new StartCrawlerRequest
        {
            Name = crawlerName,
        };

        var response = await _amazonGlue.StartCrawlerAsync(crawlerRequest);

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Start an AWS Glue job run.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <returns>A string representing the job run Id.</returns>
    public async Task<string> StartJobRunAsync(
        string jobName,
        string inputDatabase,
        string inputTable,
        string bucketName)
    {
        var request = new StartJobRunRequest
        {
            JobName = jobName,
            Arguments = new Dictionary<string, string>
            {
                {"--input_database", inputDatabase},
                {"--input_table", inputTable},
                {"--output_bucket_url", $"s3://{bucketName}/"}
            }
        };

        var response = await _amazonGlue.StartJobRunAsync(request);
        return response.JobRunId;
    }

}
```

シナリオを実行するクラスを作成します。

```
global using Amazon.Glue;
global using GlueActions;
global using Microsoft.Extensions.Configuration;
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.Glue.Model;
using Amazon.S3;
using Amazon.S3.Model;

namespace GlueBasics;

public class GlueBasics
{
    private static ILogger logger = null!;
    private static IConfiguration _configuration = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for AWS Glue.
        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<IAmazonGlue>()
            .AddTransient<GlueWrapper>()
            .AddTransient<UiWrapper>()
            )
            .Build();
```

```
        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
        .CreateLogger<GlueBasics>();

        _configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally load local settings.
            .Build();

        // These values are stored in settings.json
        // Once you have run the CDK script to deploy the resources,
        // edit the file to set "BucketName", "RoleName", and "ScriptURL"
        // to the appropriate values. Also set "CrawlerName" to the name
        // you want to give the crawler when it is created.
        string bucketName = _configuration["BucketName"]!;
        string bucketUrl = _configuration["BucketUrl"]!;
        string crawlerName = _configuration["CrawlerName"]!;
        string roleName = _configuration["RoleName"]!;
        string sourceData = _configuration["SourceData"]!;
        string dbName = _configuration["DbName"]!;
        string cron = _configuration["Cron"]!;
        string scriptUrl = _configuration["ScriptURL"]!;
        string jobName = _configuration["JobName"]!;

        var wrapper = host.Services.GetRequiredService<GlueWrapper>();
        var uiWrapper = host.Services.GetRequiredService<UiWrapper>();

        uiWrapper.DisplayOverview();
        uiWrapper.PressEnter();

        // Create the crawler and wait for it to be ready.
        uiWrapper.DisplayTitle("Create AWS Glue crawler");
        Console.WriteLine("Let's begin by creating the AWS Glue crawler.");

        var crawlerDescription = "Crawler created for the AWS Glue Basics
scenario.";
        var crawlerCreated = await wrapper.CreateCrawlerAsync(crawlerName,
crawlerDescription, roleName, cron, sourceData, dbName);
        if (crawlerCreated)
        {
            Console.WriteLine($"The crawler: {crawlerName} has been created. Now
let's wait until it's ready.");
            CrawlerState crawlerState;
```

```
            do
            {
                crawlerState = await wrapper.GetCrawlerStateAsync(crawlerName);
            }
            while (crawlerState != "READY");
            Console.WriteLine($"The crawler {crawlerName} is now ready for use.");
        }
        else
        {
            Console.WriteLine($"Couldn't create crawler {crawlerName}.");
            return; // Exit the application.
        }

        uiWrapper.DisplayTitle("Start AWS Glue crawler");
        Console.WriteLine("Now let's wait until the crawler has successfully
started.");
        var crawlerStarted = await wrapper.StartCrawlerAsync(crawlerName);
        if (crawlerStarted)
        {
            CrawlerState crawlerState;
            do
            {
                crawlerState = await wrapper.GetCrawlerStateAsync(crawlerName);
            }
            while (crawlerState != "READY");
            Console.WriteLine($"The crawler {crawlerName} is now ready for use.");
        }
        else
        {
            Console.WriteLine($"Couldn't start the crawler {crawlerName}.");
            return; // Exit the application.
        }

        uiWrapper.PressEnter();

        Console.WriteLine($"\nLet's take a look at the database: {dbName}");
        var database = await wrapper.GetDatabaseAsync(dbName);

        if (database != null)
        {
            uiWrapper.DisplayTitle($"{database.Name} Details");
            Console.WriteLine($"{database.Name} created on {database.CreateTime}");
            Console.WriteLine(database.Description);
        }
```

```
        uiWrapper.PressEnter();

        var tables = await wrapper.GetTablesAsync(dbName);
        if (tables.Count > 0)
        {
            tables.ForEach(table =>
            {
                Console.WriteLine($"{table.Name}\tCreated:
{table.CreateTime}\tUpdated: {table.UpdateTime}");
            });
        }

        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Create AWS Glue job");
        Console.WriteLine("Creating a new AWS Glue job.");
        var description = "An AWS Glue job created using the AWS SDK for .NET";
        await wrapper.CreateJobAsync(dbName, tables[0].Name, bucketUrl, jobName,
roleName, description, scriptUrl);

        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Starting AWS Glue job");
        Console.WriteLine("Starting the new AWS Glue job...");
        var jobRunId = await wrapper.StartJobRunAsync(jobName, dbName,
tables[0].Name, bucketName);
        var jobRunComplete = false;
        var jobRun = new JobRun();
        do
        {
            jobRun = await wrapper.GetJobRunAsync(jobName, jobRunId);
            if (jobRun.JobRunState == "SUCCEEDED" || jobRun.JobRunState == "STOPPED"
||
                jobRun.JobRunState == "FAILED" || jobRun.JobRunState == "TIMEOUT")
            {
                jobRunComplete = true;
            }
        } while (!jobRunComplete);

        uiWrapper.DisplayTitle($"Data in {bucketName}");

        // Get the list of data stored in the S3 bucket.
        var s3Client = new AmazonS3Client();
```

```
        var response = await s3Client.ListObjectsAsync(new ListObjectsRequest
{ BucketName = bucketName });
        response.S3Objects.ForEach(s3Object =>
        {
            Console.WriteLine(s3Object.Key);
        });

        uiWrapper.DisplayTitle("AWS Glue jobs");
        var jobNames = await wrapper.ListJobsAsync();
        jobNames.ForEach(jobName =>
        {
            Console.WriteLine(jobName);
        });

        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Get AWS Glue job run information");
        Console.WriteLine("Getting information about the AWS Glue job.");
        var jobRuns = await wrapper.GetJobRunsAsync(jobName);

        jobRuns.ForEach(jobRun =>
        {

Console.WriteLine($"{jobRun.JobName}\t{jobRun.JobRunState}\t{jobRun.CompletedOn}");
        });

        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Deleting resources");
        Console.WriteLine("Deleting the AWS Glue job used by the example.");
        await wrapper.DeleteJobAsync(jobName);

        Console.WriteLine("Deleting the tables from the database.");
        tables.ForEach(async table =>
        {
            await wrapper.DeleteTableAsync(dbName, table.Name);
        });

        Console.WriteLine("Deleting the database.");
        await wrapper.DeleteDatabaseAsync(dbName);

        Console.WriteLine("Deleting the AWS Glue crawler.");
        await wrapper.DeleteCrawlerAsync(crawlerName);
```

```
        Console.WriteLine("The AWS Glue scenario has completed.");
        uiWrapper.PressEnter();
    }
}


namespace GlueBasics;

public class UiWrapper
{
    public readonly string SepBar = new string('-', Console.WindowWidth);

    /// <summary>
    /// Show information about the scenario.
    /// </summary>
    public void DisplayOverview()
    {
        Console.Clear();
        DisplayTitle("Amazon Glue: get started with crawlers and jobs");

        Console.WriteLine("This example application does the following:");
        Console.WriteLine("\t 1. Create a crawler, pass it the IAM role and the URL
 to the public S3 bucket that contains the source data");
        Console.WriteLine("\t 2. Start the crawler.");
        Console.WriteLine("\t 3. Get the database created by the crawler and the
 tables in the database.");
        Console.WriteLine("\t 4. Create a job.");
        Console.WriteLine("\t 5. Start a job run.");
        Console.WriteLine("\t 6. Wait for the job run to complete.");
        Console.WriteLine("\t 7. Show the data stored in the bucket.");
        Console.WriteLine("\t 8. List jobs for the account.");
        Console.WriteLine("\t 9. Get job run details for the job that was run.");
        Console.WriteLine("\t10. Delete the demo job.");
        Console.WriteLine("\t11. Delete the database and tables created for the
 demo.");
        Console.WriteLine("\t12. Delete the crawler.");
    }

    /// <summary>
    /// Display a message and wait until the user presses enter.
    /// </summary>
    public void PressEnter()
    {
```

```
        Console.Write("\nPlease press <Enter> to continue. ");
        _ = Console.ReadLine();
    }

    /// <summary>
    /// Pad a string with spaces to center it on the console display.
    /// </summary>
    /// <param name="strToCenter">The string to center on the screen.</param>
    /// <returns>The string padded to make it center on the screen.</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);
    }
}
```

- 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。
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob
  - DeleteTable
  - GetCrawler

- GetDatabase
- GetDatabases
- GetJob
- GetJobRun
- GetJobRuns
- GetTables
- ListJobs
- StartCrawler
- StartJobRun

を使用した IAM の例 AWS SDK for .NET

次のコード例は、IAM AWS SDK for .NET で 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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。このリンクには GitHub、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が記載されています。

開始方法

IAM へようこそ

次のコード例は、IAM の使用を開始する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
namespace IAMActions;

public class HelloIAM
{
    static async Task Main(string[] args)
    {
        // Getting started with AWS Identity and Access Management (IAM). List
        // the policies for the account.
        var iamClient = new AmazonIdentityManagementServiceClient();

        var listPoliciesPaginator = iamClient.Paginators.ListPolicies(new
 ListPoliciesRequest());
        var policies = new List<ManagedPolicy>();

        await foreach (var response in listPoliciesPaginator.Responses)
        {
            policies.AddRange(response.Policies);
        }

        Console.WriteLine("Here are the policies defined for your account:\n");
        policies.ForEach(policy =>
        {
            Console.WriteLine($"Created:
 {policy.CreateDate}\t{policy.PolicyName}\t{policy.Description}");
        });
    }
}
```

- API の詳細については、「 API リファレンスListPolicies」の「」を参照してください。 AWS SDK for .NET

トピック

- アクション
- シナリオ

アクション

`AddUserToGroup`

次の例は、AddUserToGroup を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Add an existing IAM user to an existing IAM group.
    /// </summary>
    /// <param name="userName">The username of the user to add.</param>
    /// <param name="groupName">The name of the group to add the user to.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AddUserToGroupAsync(string userName, string groupName)
    {
        var response = await _IAMService.AddUserToGroupAsync(new
AddUserToGroupRequest
        {
            GroupName = groupName,
            UserName = userName,
        });

        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスAddUserToGroup」の「」を参照してください。 AWS SDK for .NET

`AttachRolePolicy`

次の例は、AttachRolePolicy を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Attach an IAM policy to a role.
    /// </summary>
    /// <param name="policyArn">The policy to attach.</param>
    /// <param name="roleName">The role that the policy will be attached to.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AttachRolePolicyAsync(string policyArn, string roleName)
    {
        var response = await _IAMService.AttachRolePolicyAsync(new
AttachRolePolicyRequest
        {
            PolicyArn = policyArn,
            RoleName = roleName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスAttachRolePolicy」の「」を参照してください。AWS SDK for .NET

`CreateAccessKey`

次の例は、`CreateAccessKey` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Create an IAM access key for a user.
    /// </summary>
    /// <param name="userName">The username for which to create the IAM access
    /// key.</param>
    /// <returns>The AccessKey.</returns>
    public async Task<AccessKey> CreateAccessKeyAsync(string userName)
    {
        var response = await _IAMService.CreateAccessKeyAsync(new
CreateAccessKeyRequest
        {
            UserName = userName,
        });

        return response.AccessKey;

    }
```

- API の詳細については、「 API リファレンスCreateAccessKey」の「」を参照してください。AWS SDK for .NET

`CreateGroup`

次の例は、CreateGroup を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Create an IAM group.
    /// </summary>
    /// <param name="groupName">The name to give the IAM group.</param>
    /// <returns>The IAM group that was created.</returns>
    public async Task<Group> CreateGroupAsync(string groupName)
    {
        var response = await _IAMService.CreateGroupAsync(new CreateGroupRequest
{ GroupName = groupName });
        return response.Group;
    }
```

- API の詳細については、「 API リファレンスCreateGroup」の「」を参照してください。 AWS SDK for .NET

`CreateInstanceProfile`

次の例は、`CreateInstanceProfile` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Create a policy, role, and profile that is associated with instances with a
specified name.
```

```
   /// An instance's associated profile defines a role that is assumed by the
   /// instance.The role has attached policies that specify the AWS permissions
granted to
   /// clients that run on the instance.
   /// </summary>
   /// <param name="policyName">Name to use for the policy.</param>
   /// <param name="roleName">Name to use for the role.</param>
   /// <param name="profileName">Name to use for the profile.</param>
   /// <param name="ssmOnlyPolicyFile">Path to a policy file for SSM.</param>
   /// <param name="awsManagedPolicies">AWS Managed policies to be attached to the
role.</param>
   /// <returns>The Arn of the profile.</returns>
   public async Task<string> CreateInstanceProfileWithName(
       string policyName,
       string roleName,
       string profileName,
       string ssmOnlyPolicyFile,
       List<string>? awsManagedPolicies = null)
   {

       var assumeRoleDoc = "{" +
                                "\"Version\": \"2012-10-17\"," +
                                "\"Statement\": [{" +
                                     "\"Effect\": \"Allow\"," +
                                     "\"Principal\": {" +
                                     "\"Service\": [" +
                                         "\"ec2.amazonaws.com\"" +
                                     "]" +
                                     "}," +
                                "\"Action\": \"sts:AssumeRole\"" +
                                "}]" +
                           "}";

       var policyDocument = await File.ReadAllTextAsync(ssmOnlyPolicyFile);

       var policyArn = "";

       try
       {
           var createPolicyResult = await _amazonIam.CreatePolicyAsync(
               new CreatePolicyRequest
               {
                   PolicyName = policyName,
                   PolicyDocument = policyDocument
```

```
                });
            policyArn = createPolicyResult.Policy.Arn;
        }
        catch (EntityAlreadyExistsException)
        {
            // The policy already exists, so we look it up to get the Arn.
            var policiesPaginator = _amazonIam.Paginators.ListPolicies(
                new ListPoliciesRequest()
                {
                    Scope = PolicyScopeType.Local
                });
            // Get the entire list using the paginator.
            await foreach (var policy in policiesPaginator.Policies)
            {
                if (policy.PolicyName.Equals(policyName))
                {
                    policyArn = policy.Arn;
                }
            }

            if (policyArn == null)
            {
                throw new InvalidOperationException("Policy not found");
            }
        }

        try
        {
            await _amazonIam.CreateRoleAsync(new CreateRoleRequest()
            {
                RoleName = roleName,
                AssumeRolePolicyDocument = assumeRoleDoc,
            });
            await _amazonIam.AttachRolePolicyAsync(new AttachRolePolicyRequest()
            {
                RoleName = roleName,
                PolicyArn = policyArn
            });
            if (awsManagedPolicies != null)
            {
                foreach (var awsPolicy in awsManagedPolicies)
                {
                    await _amazonIam.AttachRolePolicyAsync(new
AttachRolePolicyRequest()
```

```
                {
                    PolicyArn = $"arn:aws:iam::aws:policy/{awsPolicy}",
                    RoleName = roleName
                });
            }
        }
    }
    catch (EntityAlreadyExistsException)
    {
        Console.WriteLine("Role already exists.");
    }

    string profileArn = "";
    try
    {
        var profileCreateResponse = await _amazonIam.CreateInstanceProfileAsync(
            new CreateInstanceProfileRequest()
            {
                InstanceProfileName = profileName
            });
        // Allow time for the profile to be ready.
        profileArn = profileCreateResponse.InstanceProfile.Arn;
        Thread.Sleep(10000);
        await _amazonIam.AddRoleToInstanceProfileAsync(
            new AddRoleToInstanceProfileRequest()
            {
                InstanceProfileName = profileName,
                RoleName = roleName
            });

    }
    catch (EntityAlreadyExistsException)
    {
        Console.WriteLine("Policy already exists.");
        var profileGetResponse = await _amazonIam.GetInstanceProfileAsync(
            new GetInstanceProfileRequest()
            {
                InstanceProfileName = profileName
            });
        profileArn = profileGetResponse.InstanceProfile.Arn;
    }
    return profileArn;
}
```

- API の詳細については、「 API リファレンスCreateInstanceProfile」の「」を参照してください。 AWS SDK for .NET

`CreatePolicy`

次の例は、`CreatePolicy` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Create an IAM policy.
    /// </summary>
    /// <param name="policyName">The name to give the new IAM policy.</param>
    /// <param name="policyDocument">The policy document for the new policy.</param>
    /// <returns>The new IAM policy object.</returns>
    public async Task<ManagedPolicy> CreatePolicyAsync(string policyName, string
policyDocument)
    {
        var response = await _IAMService.CreatePolicyAsync(new CreatePolicyRequest
        {
            PolicyDocument = policyDocument,
            PolicyName = policyName,
        });

        return response.Policy;
    }
```

- API の詳細については、「 API リファレンスCreatePolicy」の「」を参照してください。 AWS SDK for .NET

CreateRole

次の例は、CreateRole を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Create a new IAM role.
    /// </summary>
    /// <param name="roleName">The name of the IAM role.</param>
    /// <param name="rolePolicyDocument">The name of the IAM policy document
    /// for the new role.</param>
    /// <returns>The Amazon Resource Name (ARN) of the role.</returns>
    public async Task<string> CreateRoleAsync(string roleName, string
rolePolicyDocument)
    {
        var request = new CreateRoleRequest
        {
            RoleName = roleName,
            AssumeRolePolicyDocument = rolePolicyDocument,
        };

        var response = await _IAMService.CreateRoleAsync(request);
        return response.Role.Arn;
    }
```

- API の詳細については、「 API リファレンスCreateRole」の「」を参照してください。 AWS SDK for .NET

CreateServiceLinkedRole

次の例は、CreateServiceLinkedRole を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Create an IAM service-linked role.
    /// </summary>
    /// <param name="serviceName">The name of the AWS Service.</param>
    /// <param name="description">A description of the IAM service-linked role.</
param>
    /// <returns>The IAM role that was created.</returns>
    public async Task<Role> CreateServiceLinkedRoleAsync(string serviceName, string
 description)
    {
        var request = new CreateServiceLinkedRoleRequest
        {
            AWSServiceName = serviceName,
            Description = description
        };

        var response = await _IAMService.CreateServiceLinkedRoleAsync(request);
        return response.Role;
    }
```

- API の詳細については、「 API リファレンスCreateServiceLinkedRole」の「」を参照してください。 AWS SDK for .NET

`CreateUser`

次の例は、`CreateUser` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Create an IAM user.
    /// </summary>
    /// <param name="userName">The username for the new IAM user.</param>
    /// <returns>The IAM user that was created.</returns>
    public async Task<User> CreateUserAsync(string userName)
    {
        var response = await _IAMService.CreateUserAsync(new CreateUserRequest
{ UserName = userName });
        return response.User;
    }
```

- API の詳細については、「 API リファレンスCreateUser」の「」を参照してください。 AWS SDK for .NET

`DeleteAccessKey`

次の例は、DeleteAccessKey を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete an IAM user's access key.
    /// </summary>
```

```
    /// <param name="accessKeyId">The Id for the IAM access key.</param>
    /// <param name="userName">The username of the user that owns the IAM
    /// access key.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteAccessKeyAsync(string accessKeyId, string
userName)
    {
        var response = await _IAMService.DeleteAccessKeyAsync(new
DeleteAccessKeyRequest
        {
            AccessKeyId = accessKeyId,
            UserName = userName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスDeleteAccessKey」の「」を参照してください。 AWS SDK for .NET

`DeleteGroup`

次の例は、DeleteGroup を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete an IAM group.
    /// </summary>
    /// <param name="groupName">The name of the IAM group to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteGroupAsync(string groupName)
    {
```

```
        var response = await _IAMService.DeleteGroupAsync(new DeleteGroupRequest
{ GroupName = groupName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスDeleteGroup」の「」を参照してください。 AWS SDK for .NET

`DeleteGroupPolicy`

次の例は、DeleteGroupPolicy を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete an IAM policy associated with an IAM group.
    /// </summary>
    /// <param name="groupName">The name of the IAM group associated with the
    /// policy.</param>
    /// <param name="policyName">The name of the policy to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteGroupPolicyAsync(string groupName, string
policyName)
    {
        var request = new DeleteGroupPolicyRequest()
        {
            GroupName = groupName,
            PolicyName = policyName,
        };

        var response = await _IAMService.DeleteGroupPolicyAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスDeleteGroupPolicy」の「」を参照してください。AWS SDK for .NET

`DeleteInstanceProfile`

次の例は、`DeleteInstanceProfile` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Detaches a role from an instance profile, detaches policies from the role,
    /// and deletes all the resources.
    /// </summary>
    /// <param name="profileName">The name of the profile to delete.</param>
    /// <param name="roleName">The name of the role to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteInstanceProfile(string profileName, string roleName)
    {
        try
        {
            await _amazonIam.RemoveRoleFromInstanceProfileAsync(
                new RemoveRoleFromInstanceProfileRequest()
                {
                    InstanceProfileName = profileName,
                    RoleName = roleName
                });
            await _amazonIam.DeleteInstanceProfileAsync(
                new DeleteInstanceProfileRequest() { InstanceProfileName =
profileName });
            var attachedPolicies = await _amazonIam.ListAttachedRolePoliciesAsync(
                new ListAttachedRolePoliciesRequest() { RoleName = roleName });
            foreach (var policy in attachedPolicies.AttachedPolicies)
```

```
            {
                await _amazonIam.DetachRolePolicyAsync(
                    new DetachRolePolicyRequest()
                    {
                        RoleName = roleName,
                        PolicyArn = policy.PolicyArn
                    });
                // Delete the custom policies only.
                if (!policy.PolicyArn.StartsWith("arn:aws:iam::aws"))
                {
                    await _amazonIam.DeletePolicyAsync(
                        new Amazon.IdentityManagement.Model.DeletePolicyRequest()
                        {
                            PolicyArn = policy.PolicyArn
                        });
                }
            }

            await _amazonIam.DeleteRoleAsync(
                new DeleteRoleRequest() { RoleName = roleName });
        }
        catch (NoSuchEntityException)
        {
            Console.WriteLine($"Instance profile {profileName} does not exist.");
        }
    }
```

- API の詳細については、「 API リファレンスDeleteInstanceProfile」の「」を参照してください。 AWS SDK for .NET

`DeletePolicy`

次の例は、DeletePolicy を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete an IAM policy.
    /// </summary>
    /// <param name="policyArn">The Amazon Resource Name (ARN) of the policy to
    /// delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeletePolicyAsync(string policyArn)
    {
        var response = await _IAMService.DeletePolicyAsync(new DeletePolicyRequest
{ PolicyArn = policyArn });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスDeletePolicy」の「」を参照してください。 AWS SDK for .NET

`DeleteRole`

次の例は、DeleteRole を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete an IAM role.
    /// </summary>
    /// <param name="roleName">The name of the IAM role to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteRoleAsync(string roleName)
    {
        var response = await _IAMService.DeleteRoleAsync(new DeleteRoleRequest
{ RoleName = roleName });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスDeleteRole」の「」を参照してください。 AWS SDK for .NET

`DeleteRolePolicy`

次の例は、`DeleteRolePolicy` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
   /// <summary>
   /// Delete an IAM role policy.
   /// </summary>
   /// <param name="roleName">The name of the IAM role.</param>
   /// <param name="policyName">The name of the IAM role policy to delete.</param>
   /// <returns>A Boolean value indicating the success of the action.</returns>
   public async Task<bool> DeleteRolePolicyAsync(string roleName, string
policyName)
   {
       var response = await _IAMService.DeleteRolePolicyAsync(new
DeleteRolePolicyRequest
       {
           PolicyName = policyName,
           RoleName = roleName,
       });

       return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
   }
```

- API の詳細については、「 API リファレンスDeleteRolePolicy」の「」を参照してください。 AWS SDK for .NET

DeleteUser

次の例は、DeleteUser を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete an IAM user.
    /// </summary>
    /// <param name="userName">The username of the IAM user to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteUserAsync(string userName)
    {
        var response = await _IAMService.DeleteUserAsync(new DeleteUserRequest
{ UserName = userName });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスDeleteUser」の「」を参照してください。 AWS SDK for .NET

DeleteUserPolicy

次の例は、DeleteUserPolicy を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete an IAM user policy.
    /// </summary>
    /// <param name="policyName">The name of the IAM policy to delete.</param>
    /// <param name="userName">The username of the IAM user.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteUserPolicyAsync(string policyName, string
 userName)
    {
        var response = await _IAMService.DeleteUserPolicyAsync(new
 DeleteUserPolicyRequest { PolicyName = policyName, UserName = userName });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスDeleteUserPolicy」の「」を参照してください。 AWS SDK for .NET

`DetachRolePolicy`

次の例は、DetachRolePolicy を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Detach an IAM policy from an IAM role.
    /// </summary>
    /// <param name="policyArn">The Amazon Resource Name (ARN) of the IAM policy.</
param>
    /// <param name="roleName">The name of the IAM role.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DetachRolePolicyAsync(string policyArn, string roleName)
    {
```

```
        var response = await _IAMService.DetachRolePolicyAsync(new
DetachRolePolicyRequest
        {
            PolicyArn = policyArn,
            RoleName = roleName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスDetachRolePolicy」の「」を参照してください。AWS SDK for .NET

`GetAccountPasswordPolicy`

次の例は、`GetAccountPasswordPolicy` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Gets the IAM password policy for an AWS account.
    /// </summary>
    /// <returns>The PasswordPolicy for the AWS account.</returns>
    public async Task<PasswordPolicy> GetAccountPasswordPolicyAsync()
    {
        var response = await _IAMService.GetAccountPasswordPolicyAsync(new
GetAccountPasswordPolicyRequest());
        return response.PasswordPolicy;
    }
```

- API の詳細については、「 API リファレンスGetAccountPasswordPolicy」の「」を参照してください。 AWS SDK for .NET

GetPolicy

次の例は、GetPolicy を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get information about an IAM policy.
    /// </summary>
    /// <param name="policyArn">The IAM policy to retrieve information for.</param>
    /// <returns>The IAM policy.</returns>
    public async Task<ManagedPolicy> GetPolicyAsync(string policyArn)
    {

        var response = await _IAMService.GetPolicyAsync(new GetPolicyRequest
{ PolicyArn = policyArn });
        return response.Policy;
    }
```

- API の詳細については、「 API リファレンスGetPolicy」の「」を参照してください。 AWS SDK for .NET

GetRole

次の例は、GetRole を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
/// Get information about an IAM role.
/// </summary>
/// <param name="roleName">The name of the IAM role to retrieve information
/// for.</param>
/// <returns>The IAM role that was retrieved.</returns>
public async Task<Role> GetRoleAsync(string roleName)
{
    var response = await _IAMService.GetRoleAsync(new GetRoleRequest
    {
        RoleName = roleName,
    });

    return response.Role;
}
```

- API の詳細については、「 API リファレンスGetRole」の「」を参照してください。 AWS SDK for .NET

`GetUser`

次の例は、`GetUser` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get information about an IAM user.
    /// </summary>
    /// <param name="userName">The username of the user.</param>
    /// <returns>An IAM user object.</returns>
    public async Task<User> GetUserAsync(string userName)
    {
        var response = await _IAMService.GetUserAsync(new GetUserRequest { UserName
= userName });
        return response.User;
    }
```

- API の詳細については、「 API リファレンスGetUser」の「」を参照してください。 AWS SDK for .NET

`ListAttachedRolePolicies`

次の例は、`ListAttachedRolePolicies` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// List the IAM role policies that are attached to an IAM role.
    /// </summary>
    /// <param name="roleName">The IAM role to list IAM policies for.</param>
    /// <returns>A list of the IAM policies attached to the IAM role.</returns>
    public async Task<List<AttachedPolicyType>> ListAttachedRolePoliciesAsync(string
roleName)
    {
        var attachedPolicies = new List<AttachedPolicyType>();
        var attachedRolePoliciesPaginator =
_IAMService.Paginators.ListAttachedRolePolicies(new ListAttachedRolePoliciesRequest
{ RoleName = roleName });
```

```
        await foreach (var response in attachedRolePoliciesPaginator.Responses)
        {
            attachedPolicies.AddRange(response.AttachedPolicies);
        }

        return attachedPolicies;
    }
```

- API の詳細については、「 API リファレンスListAttachedRolePolicies」の「」を参照してください。 AWS SDK for .NET

`ListGroups`

次の例は、`ListGroups` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// List IAM groups.
    /// </summary>
    /// <returns>A list of IAM groups.</returns>
    public async Task<List<Group>> ListGroupsAsync()
    {
        var groupsPaginator = _IAMService.Paginators.ListGroups(new
ListGroupsRequest());
        var groups = new List<Group>();

        await foreach (var response in groupsPaginator.Responses)
        {
            groups.AddRange(response.Groups);
        }
```

```
        return groups;
    }
```

- API の詳細については、「 API リファレンスListGroups」の「」を参照してください。 AWS SDK for .NET

`ListPolicies`

次の例は、`ListPolicies` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// List IAM policies.
    /// </summary>
    /// <returns>A list of the IAM policies.</returns>
    public async Task<List<ManagedPolicy>> ListPoliciesAsync()
    {
        var listPoliciesPaginator = _IAMService.Paginators.ListPolicies(new
ListPoliciesRequest());
        var policies = new List<ManagedPolicy>();

        await foreach (var response in listPoliciesPaginator.Responses)
        {
            policies.AddRange(response.Policies);
        }

        return policies;
    }
```

- API の詳細については、「 API リファレンスListPolicies」の「」を参照してください。 AWS SDK for .NET

`ListRolePolicies`

次の例は、`ListRolePolicies` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// List IAM role policies.
    /// </summary>
    /// <param name="roleName">The IAM role for which to list IAM policies.</param>
    /// <returns>A list of IAM policy names.</returns>
    public async Task<List<string>> ListRolePoliciesAsync(string roleName)
    {
        var listRolePoliciesPaginator = _IAMService.Paginators.ListRolePolicies(new
ListRolePoliciesRequest { RoleName = roleName });
        var policyNames = new List<string>();

        await foreach (var response in listRolePoliciesPaginator.Responses)
        {
            policyNames.AddRange(response.PolicyNames);
        }

        return policyNames;
    }
```

- API の詳細については、「 API リファレンスListRolePolicies」の「」を参照してください。 AWS SDK for .NET

ListRoles

次の例は、ListRoles を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// List IAM roles.
    /// </summary>
    /// <returns>A list of IAM roles.</returns>
    public async Task<List<Role>> ListRolesAsync()
    {
        var listRolesPaginator = _IAMService.Paginators.ListRoles(new
ListRolesRequest());
        var roles = new List<Role>();

        await foreach (var response in listRolesPaginator.Responses)
        {
            roles.AddRange(response.Roles);
        }

        return roles;
    }
```

- API の詳細については、「 API リファレンスListRoles」の「」を参照してください。 AWS SDK for .NET

ListSAMLProviders

次の例は、ListSAMLProviders を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、AWS コードサンプル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 SAML authentication providers.
    /// </summary>
    /// <returns>A list of SAML providers.</returns>
    public async Task<List<SAMLProviderListEntry>> ListSAMLProvidersAsync()
    {
        var response = await _IAMService.ListSAMLProvidersAsync(new
ListSAMLProvidersRequest());
        return response.SAMLProviderList;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「ListSAMLProviders」を参照してください。

`ListUsers`

次の例は、`ListUsers` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// List IAM users.
    /// </summary>
    /// <returns>A list of IAM users.</returns>
```

```
    public async Task<List<User>> ListUsersAsync()
    {
        var listUsersPaginator = _IAMService.Paginators.ListUsers(new
 ListUsersRequest());
        var users = new List<User>();

        await foreach (var response in listUsersPaginator.Responses)
        {
            users.AddRange(response.Users);
        }

        return users;
    }
```

- API の詳細については、「 API リファレンスListUsers」の「」を参照してください。 AWS SDK for .NET

PutGroupPolicy

次の例は、PutGroupPolicy を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Add or update an inline policy document that is embedded in an IAM group.
    /// </summary>
    /// <param name="groupName">The name of the IAM group.</param>
    /// <param name="policyName">The name of the IAM policy.</param>
    /// <param name="policyDocument">The policy document defining the IAM policy.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> PutGroupPolicyAsync(string groupName, string policyName,
 string policyDocument)
```

```
    {
        var request = new PutGroupPolicyRequest
        {
            GroupName = groupName,
            PolicyName = policyName,
            PolicyDocument = policyDocument
        };

        var response = await _IAMService.PutGroupPolicyAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスPutGroupPolicy」の「」を参照してください。 AWS SDK for .NET

`PutRolePolicy`

次の例は、PutRolePolicy を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Update the inline policy document embedded in a role.
    /// </summary>
    /// <param name="policyName">The name of the policy to embed.</param>
    /// <param name="roleName">The name of the role to update.</param>
    /// <param name="policyDocument">The policy document that defines the role.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> PutRolePolicyAsync(string policyName, string roleName,
 string policyDocument)
    {
        var request = new PutRolePolicyRequest
```

```
        {
            PolicyName = policyName,
            RoleName = roleName,
            PolicyDocument = policyDocument
        };

        var response = await _IAMService.PutRolePolicyAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスPutRolePolicy」の「」を参照してください。 AWS SDK for .NET

`RemoveUserFromGroup`

次の例は、RemoveUserFromGroup を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Remove a user from an IAM group.
    /// </summary>
    /// <param name="userName">The username of the user to remove.</param>
    /// <param name="groupName">The name of the IAM group to remove the user from.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> RemoveUserFromGroupAsync(string userName, string
 groupName)
    {
        // Remove the user from the group.
        var removeUserRequest = new RemoveUserFromGroupRequest()
        {
            UserName = userName,
```

```
            GroupName = groupName,
        };

        var response = await
_IAMService.RemoveUserFromGroupAsync(removeUserRequest);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスRemoveUserFromGroup」の「」を参照してください。 AWS SDK for .NET

シナリオ

レジリエントなサービスの構築と管理

次のコード例は、本、映画、曲のレコメンデーションを返す負荷分散型ウェブサービスの作成方法を示しています。この例は、障害に対するサービスの対応方法と、障害発生時の耐障害性を高めるためにサービスを再構築する方法を示しています。

- Amazon EC2 Auto Scaling グループを使用して、起動テンプレートに基づいて Amazon Elastic Compute Cloud (Amazon EC2) インスタンスを作成し、インスタンス数を所定の範囲内に維持します。
- Elastic Load Balancing で HTTP リクエストを処理して配信します。
- Auto Scaling グループ内のインスタンスの状態を監視し、正常なインスタンスにのみリクエストを転送します。
- 各 EC2 インスタンスで Python ウェブサーバーを実行して HTTP リクエストを処理します。ウェブサーバーはレコメンデーションとヘルスチェックを返します。
- Amazon DynamoDB テーブルを使用してレコメンデーションサービスをシミュレートできます。
- AWS Systems Manager パラメータを更新して、リクエストとヘルスチェックに対するウェブサーバーの応答を制御します。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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のシナリオを実行します。

```
    static async Task Main(string[] args)
    {
        _configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally, load local settings.
            .Build();


        // Set up dependency injection for the AWS services.
        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<IAmazonIdentityManagementService>()
                    .AddAWSService<IAmazonDynamoDB>()
                    .AddAWSService<IAmazonElasticLoadBalancingV2>()
                    .AddAWSService<IAmazonSimpleSystemsManagement>()
                    .AddAWSService<IAmazonAutoScaling>()
                    .AddAWSService<IAmazonEC2>()
                    .AddTransient<AutoScalerWrapper>()
                    .AddTransient<ElasticLoadBalancerWrapper>()
                    .AddTransient<SmParameterWrapper>()
                    .AddTransient<Recommendations>()
                    .AddSingleton<IConfiguration>(_configuration)
            )
            .Build();

        ServicesSetup(host);
```

```
        ResourcesSetup();

        try
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Welcome to the Resilient Architecture Example
Scenario.");
            Console.WriteLine(new string('-', 80));
            await Deploy(true);

            Console.WriteLine("Now let's begin the scenario.");
            Console.WriteLine(new string('-', 80));
            await Demo(true);

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Finally, let's clean up our resources.");
            Console.WriteLine(new string('-', 80));

            await DestroyResources(true);

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Resilient Architecture Example Scenario is
complete.");
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
            await DestroyResources(true);
            Console.WriteLine(new string('-', 80));
        }
    }

    /// <summary>
    /// Setup any common resources, also used for integration testing.
    /// </summary>
    public static void ResourcesSetup()
    {
        _httpClient = new HttpClient();
    }

    /// <summary>
```

```
    /// Populate the services for use within the console application.
    /// </summary>
    /// <param name="host">The services host.</param>
    private static void ServicesSetup(IHost host)
    {
        _elasticLoadBalancerWrapper =
host.Services.GetRequiredService<ElasticLoadBalancerWrapper>();
        _iamClient =
host.Services.GetRequiredService<IAmazonIdentityManagementService>();
        _recommendations = host.Services.GetRequiredService<Recommendations>();
        _autoScalerWrapper = host.Services.GetRequiredService<AutoScalerWrapper>();
        _smParameterWrapper =
host.Services.GetRequiredService<SmParameterWrapper>();
    }

    /// <summary>
    /// Deploy necessary resources for the scenario.
    /// </summary>
    /// <param name="interactive">True to run as interactive.</param>
    /// <returns>True if successful.</returns>
    public static async Task<bool> Deploy(bool interactive)
    {
        var protocol = "HTTP";
        var port = 80;
        var sshPort = 22;

        Console.WriteLine(
            "\nFor this demo, we'll use the AWS SDK for .NET to create several AWS
resources\n" +
            "to set up a load-balanced web service endpoint and explore some ways to
make it resilient\n" +
            "against various kinds of failures.\n\n" +
            "Some of the resources create by this demo are:\n");

        Console.WriteLine(
            "\t* A DynamoDB table that the web service depends on to provide book,
movie, and song recommendations.");
        Console.WriteLine(
            "\t* An EC2 launch template that defines EC2 instances that each contain
a Python web server.");
        Console.WriteLine(
            "\t* An EC2 Auto Scaling group that manages EC2 instances across several
Availability Zones.");
        Console.WriteLine(
```

```
            "\t* An Elastic Load Balancing (ELB) load balancer that targets the Auto
Scaling group to distribute requests.");
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to start deploying
resources.");
        if (interactive)
            Console.ReadLine();

        // Create and populate the DynamoDB table.
        var databaseTableName = _configuration["databaseName"];
        var recommendationsPath = Path.Join(_configuration["resourcePath"],
            "recommendations_objects.json");
        Console.WriteLine($"Creating and populating a DynamoDB table named
{databaseTableName}.");
        await _recommendations.CreateDatabaseWithName(databaseTableName);
        await _recommendations.PopulateDatabase(databaseTableName,
recommendationsPath);
        Console.WriteLine(new string('-', 80));

        // Create the EC2 Launch Template.

        Console.WriteLine(
            $"Creating an EC2 launch template that runs 'server_startup_script.sh'
when an instance starts.\n"
            + "\nThis script starts a Python web server defined in the `server.py`
script. The web server\n"
            + "listens to HTTP requests on port 80 and responds to requests to '/'
and to '/healthcheck'.\n"
            + "For demo purposes, this server is run as the root user. In
production, the best practice is to\n"
            + "run a web server, such as Apache, with least-privileged
credentials.");
        Console.WriteLine(
            "\nThe template also defines an IAM policy that each instance uses to
assume a role that grants\n"
            + "permissions to access the DynamoDB recommendation table and Systems
Manager parameters\n"
            + "that control the flow of the demo.");

        var startupScriptPath = Path.Join(_configuration["resourcePath"],
            "server_startup_script.sh");
        var instancePolicyPath = Path.Join(_configuration["resourcePath"],
            "instance_policy.json");
```

```
        await _autoScalerWrapper.CreateTemplate(startupScriptPath,
instancePolicyPath);
        Console.WriteLine(new string('-', 80));

        Console.WriteLine(
            "Creating an EC2 Auto Scaling group that maintains three EC2 instances,
each in a different\n"
            + "Availability Zone.\n");
        var zones = await _autoScalerWrapper.DescribeAvailabilityZones();
        await _autoScalerWrapper.CreateGroupOfSize(3, _autoScalerWrapper.GroupName,
zones);
        Console.WriteLine(new string('-', 80));

        Console.WriteLine(
            "At this point, you have EC2 instances created. Once each instance
starts, it listens for\n"
            + "HTTP requests. You can see these instances in the console or continue
with the demo.\n");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to continue.");
        if (interactive)
            Console.ReadLine();

        Console.WriteLine("Creating variables that control the flow of the demo.");
        await _smParameterWrapper.Reset();

        Console.WriteLine(
            "\nCreating an Elastic Load Balancing target group and load balancer.
The target group\n"
            + "defines how the load balancer connects to instances. The load
balancer provides a\n"
            + "single endpoint where clients connect and dispatches requests to
instances in the group.");

        var defaultVpc = await _autoScalerWrapper.GetDefaultVpc();
        var subnets = await
_autoScalerWrapper.GetAllVpcSubnetsForZones(defaultVpc.VpcId, zones);
        var subnetIds = subnets.Select(s => s.SubnetId).ToList();
        var targetGroup = await
_elasticLoadBalancerWrapper.CreateTargetGroupOnVpc(_elasticLoadBalancerWrapper.TargetGroupN
protocol, port, defaultVpc.VpcId);
```

```
        await
 _elasticLoadBalancerWrapper.CreateLoadBalancerAndListener(_elasticLoadBalancerWrapper.LoadB
 subnetIds, targetGroup);
        await
 _autoScalerWrapper.AttachLoadBalancerToGroup(_autoScalerWrapper.GroupName,
 targetGroup.TargetGroupArn);
        Console.WriteLine("\nVerifying access to the load balancer endpoint...");
        var endPoint = await
 _elasticLoadBalancerWrapper.GetEndpointForLoadBalancerByName(_elasticLoadBalancerWrapper.Lo
        var loadBalancerAccess = await
 _elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);

        if (!loadBalancerAccess)
        {
            Console.WriteLine("\nCouldn't connect to the load balancer, verifying
 that the port is open...");

            var ipString = await _httpClient.GetStringAsync("https://
checkip.amazonaws.com");
            ipString = ipString.Trim();

            var defaultSecurityGroup = await
 _autoScalerWrapper.GetDefaultSecurityGroupForVpc(defaultVpc);
            var portIsOpen =
 _autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, port, ipString);
            var sshPortIsOpen =
 _autoScalerWrapper.VerifyInboundPortForGroup(defaultSecurityGroup, sshPort,
 ipString);

            if (!portIsOpen)
            {
                Console.WriteLine(
                    "\nFor this example to work, the default security group for your
 default VPC must\n"
                    + "allows access from this computer. You can either add it
 automatically from this\n"
                    + "example or add it yourself using the AWS Management Console.
\n");

                if (!interactive || GetYesNoResponse(
                        "Do you want to add a rule to the security group to allow
 inbound traffic from your computer's IP address?"))
                {
```

```
                    await
_autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, port, ipString);
                }
            }

            if (!sshPortIsOpen)
            {
                if (!interactive || GetYesNoResponse(
                        "Do you want to add a rule to the security group to allow
inbound SSH traffic for debugging from your computer's IP address?"))
                {
                    await
_autoScalerWrapper.OpenInboundPort(defaultSecurityGroup.GroupId, sshPort,
ipString);
                }
            }
            loadBalancerAccess = await
_elasticLoadBalancerWrapper.VerifyLoadBalancerEndpoint(endPoint);
        }

        if (loadBalancerAccess)
        {
            Console.WriteLine("Your load balancer is ready. You can access it by
browsing to:");
            Console.WriteLine($"\thttp://{endPoint}\n");
        }
        else
        {
            Console.WriteLine(
                "\nCouldn't get a successful response from the load balancer
endpoint. Troubleshoot by\n"
                + "manually verifying that your VPC and security group are
configured correctly and that\n"
                + "you can successfully make a GET request to the load balancer
endpoint:\n");
            Console.WriteLine($"\thttp://{endPoint}\n");
        }
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you're ready to continue with the
demo.");
        if (interactive)
            Console.ReadLine();
        return true;
    }
```

```
    /// <summary>
    /// Demonstrate the steps of the scenario.
    /// </summary>
    /// <param name="interactive">True to run as an interactive scenario.</param>
    /// <returns>Async task.</returns>
    public static async Task<bool> Demo(bool interactive)
    {
        var ssmOnlyPolicy = Path.Join(_configuration["resourcePath"],
            "ssm_only_policy.json");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Resetting parameters to starting values for demo.");
        await _smParameterWrapper.Reset();

        Console.WriteLine("\nThis part of the demonstration shows how to toggle
 different parts of the system\n" +
                          "to create situations where the web service fails, and
 shows how using a resilient\n" +
                          "architecture can keep the web service running in spite of
 these failures.");
        Console.WriteLine(new string('-', 88));
        Console.WriteLine("At the start, the load balancer endpoint returns
 recommendations and reports that all targets are healthy.");
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine($"The web service running on the EC2 instances gets
 recommendations by querying a DynamoDB table.\n" +
                          $"The table name is contained in a Systems Manager
 parameter named '{_smParameterWrapper.TableParameter}'.\n" +
                          $"To simulate a failure of the recommendation service,
 let's set this parameter to name a non-existent table.\n");
        await
 _smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter, "this-
is-not-a-table");
        Console.WriteLine("\nNow, sending a GET request to the load balancer
 endpoint returns a failure code. But, the service reports as\n" +
                          "healthy to the load balancer because shallow health
 checks don't check for failure of the recommendation service.");
        if (interactive)
            await DemoActionChoices();
```

```
        Console.WriteLine("Instead of failing when the recommendation service fails,
the web service can return a static response.");
        Console.WriteLine("While this is not a perfect solution, it presents the
customer with a somewhat better experience than failure.");

        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.FailureResponseParameter,
"static");

        Console.WriteLine("\nNow, sending a GET request to the load balancer
endpoint returns a static response.");
        Console.WriteLine("The service still reports as healthy because health
checks are still shallow.");
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("Let's reinstate the recommendation service.\n");
        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter,
_smParameterWrapper.TableName);
        Console.WriteLine(
            "\nLet's also substitute bad credentials for one of the instances in the
target group so that it can't\n" +
            "access the DynamoDB recommendation table.\n"
        );
        await _autoScalerWrapper.CreateInstanceProfileWithName(
            _autoScalerWrapper.BadCredsPolicyName,
            _autoScalerWrapper.BadCredsRoleName,
            _autoScalerWrapper.BadCredsProfileName,
            ssmOnlyPolicy,
            new List<string> { "AmazonSSMManagedInstanceCore" }
        );
        var instances = await
_autoScalerWrapper.GetInstancesByGroupName(_autoScalerWrapper.GroupName);
        var badInstanceId = instances.First();
        var instanceProfile = await
_autoScalerWrapper.GetInstanceProfile(badInstanceId);
        Console.WriteLine(
            $"Replacing the profile for instance {badInstanceId} with a profile that
contains\n" +
            "bad credentials...\n"
        );
        await _autoScalerWrapper.ReplaceInstanceProfile(
            badInstanceId,
```

```
            _autoScalerWrapper.BadCredsProfileName,
            instanceProfile.AssociationId
        );
        Console.WriteLine(
            "Now, sending a GET request to the load balancer endpoint returns either
a recommendation or a static response,\n" +
            "depending on which instance is selected by the load balancer.\n"
        );
        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nLet's implement a deep health check. For this demo, a
deep health check tests whether");
        Console.WriteLine("the web service can access the DynamoDB table that it
depends on for recommendations. Note that");
        Console.WriteLine("the deep health check is only for ELB routing and not for
Auto Scaling instance health.");
        Console.WriteLine("This kind of deep health check is not recommended for
Auto Scaling instance health, because it");
        Console.WriteLine("risks accidental termination of all instances in the Auto
Scaling group when a dependent service fails.");

        Console.WriteLine("\nBy implementing deep health checks, the load balancer
can detect when one of the instances is failing");
        Console.WriteLine("and take that instance out of rotation.");

        await
_smParameterWrapper.PutParameterByName(_smParameterWrapper.HealthCheckParameter,
"deep");

        Console.WriteLine($"\nNow, checking target health indicates that the
instance with bad credentials ({badInstanceId})");
        Console.WriteLine("is unhealthy. Note that it might take a minute or two for
the load balancer to detect the unhealthy");
        Console.WriteLine("instance. Sending a GET request to the load balancer
endpoint always returns a recommendation, because");
        Console.WriteLine("the load balancer takes unhealthy instances out of its
rotation.");

        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nBecause the instances in this demo are controlled by an
auto scaler, the simplest way to fix an unhealthy");
```

```
        Console.WriteLine("instance is to terminate it and let the auto scaler start
 a new instance to replace it.");

        await _autoScalerWrapper.TryTerminateInstanceById(badInstanceId);

        Console.WriteLine($"\nEven while the instance is terminating and the new
 instance is starting, sending a GET");
        Console.WriteLine("request to the web service continues to get a successful
 recommendation response because");
        Console.WriteLine("starts and reports as healthy, it is included in the load
 balancing rotation.");
        Console.WriteLine("Note that terminating and replacing an instance typically
 takes several minutes, during which time you");
        Console.WriteLine("can see the changing health check status until the new
 instance is running and healthy.");

        if (interactive)
            await DemoActionChoices();

        Console.WriteLine("\nIf the recommendation service fails now, deep health
 checks mean all instances report as unhealthy.");

        await
 _smParameterWrapper.PutParameterByName(_smParameterWrapper.TableParameter, "this-
is-not-a-table");

        Console.WriteLine($"\nWhen all instances are unhealthy, the load balancer
 continues to route requests even to");
        Console.WriteLine("unhealthy instances, allowing them to fail open and
 return a static response rather than fail");
        Console.WriteLine("closed and report failure to the customer.");

        if (interactive)
            await DemoActionChoices();
        await _smParameterWrapper.Reset();

        Console.WriteLine(new string('-', 80));
        return true;
    }

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <param name="interactive">True to ask the user for cleanup.</param>
```

```
    /// <returns>Async task.</returns>
    public static async Task<bool> DestroyResources(bool interactive)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine(
            "To keep things tidy and to avoid unwanted charges on your account, we
can clean up all AWS resources\n" +
            "that were created for this demo."
        );

        if (!interactive || GetYesNoResponse("Do you want to clean up all demo
resources? (y/n) "))
        {
            await
_elasticLoadBalancerWrapper.DeleteLoadBalancerByName(_elasticLoadBalancerWrapper.LoadBalanc
            await
_elasticLoadBalancerWrapper.DeleteTargetGroupByName(_elasticLoadBalancerWrapper.TargetGroup
            await
_autoScalerWrapper.TerminateAndDeleteAutoScalingGroupWithName(_autoScalerWrapper.GroupName)
            await
_autoScalerWrapper.DeleteKeyPairByName(_autoScalerWrapper.KeyPairName);
            await
_autoScalerWrapper.DeleteTemplateByName(_autoScalerWrapper.LaunchTemplateName);
            await _autoScalerWrapper.DeleteInstanceProfile(
                _autoScalerWrapper.BadCredsProfileName,
                _autoScalerWrapper.BadCredsRoleName
            );
            await
_recommendations.DestroyDatabaseByName(_recommendations.TableName);
        }
        else
        {
            Console.WriteLine(
                "Ok, we'll leave the resources intact.\n" +
                "Don't forget to delete them when you're done with them or you might
incur unexpected charges."
            );
        }

        Console.WriteLine(new string('-', 80));
        return true;
    }
```

Auto Scaling と Amazon EC2 のアクションをラップするクラスを作成します。

```
/// <summary>
/// Encapsulates Amazon EC2 Auto Scaling and EC2 management methods.
/// </summary>
public class AutoScalerWrapper
{
    private readonly IAmazonAutoScaling _amazonAutoScaling;
    private readonly IAmazonEC2 _amazonEc2;
    private readonly IAmazonSimpleSystemsManagement _amazonSsm;
    private readonly IAmazonIdentityManagementService _amazonIam;

    private readonly string _instanceType = "";
    private readonly string _amiParam = "";
    private readonly string _launchTemplateName = "";
    private readonly string _groupName = "";
    private readonly string _instancePolicyName = "";
    private readonly string _instanceRoleName = "";
    private readonly string _instanceProfileName = "";
    private readonly string _badCredsProfileName = "";
    private readonly string _badCredsRoleName = "";
    private readonly string _badCredsPolicyName = "";
    private readonly string _keyPairName = "";

    public string GroupName => _groupName;
    public string KeyPairName => _keyPairName;
    public string LaunchTemplateName => _launchTemplateName;
    public string InstancePolicyName => _instancePolicyName;
    public string BadCredsProfileName => _badCredsProfileName;
    public string BadCredsRoleName => _badCredsRoleName;
    public string BadCredsPolicyName => _badCredsPolicyName;

    /// <summary>
    /// Constructor for the AutoScalerWrapper.
    /// </summary>
    /// <param name="amazonAutoScaling">The injected AutoScaling client.</param>
    /// <param name="amazonEc2">The injected EC2 client.</param>
    /// <param name="amazonIam">The injected IAM client.</param>
    /// <param name="amazonSsm">The injected SSM client.</param>
    public AutoScalerWrapper(
        IAmazonAutoScaling amazonAutoScaling,
        IAmazonEC2 amazonEc2,
        IAmazonSimpleSystemsManagement amazonSsm,
        IAmazonIdentityManagementService amazonIam,
```

```
        IConfiguration configuration)
    {
        _amazonAutoScaling = amazonAutoScaling;
        _amazonEc2 = amazonEc2;
        _amazonSsm = amazonSsm;
        _amazonIam = amazonIam;

        var prefix = configuration["resourcePrefix"];
        _instanceType = configuration["instanceType"];
        _amiParam = configuration["amiParam"];

        _launchTemplateName = prefix + "-template";
        _groupName = prefix + "-group";
        _instancePolicyName = prefix + "-pol";
        _instanceRoleName = prefix + "-role";
        _instanceProfileName = prefix + "-prof";
        _badCredsPolicyName = prefix + "-bc-pol";
        _badCredsRoleName = prefix + "-bc-role";
        _badCredsProfileName = prefix + "-bc-prof";
        _keyPairName = prefix + "-key-pair";
    }

    /// <summary>
    /// Create a policy, role, and profile that is associated with instances with a
specified name.
    /// An instance's associated profile defines a role that is assumed by the
    /// instance.The role has attached policies that specify the AWS permissions
granted to
    /// clients that run on the instance.
    /// </summary>
    /// <param name="policyName">Name to use for the policy.</param>
    /// <param name="roleName">Name to use for the role.</param>
    /// <param name="profileName">Name to use for the profile.</param>
    /// <param name="ssmOnlyPolicyFile">Path to a policy file for SSM.</param>
    /// <param name="awsManagedPolicies">AWS Managed policies to be attached to the
role.</param>
    /// <returns>The Arn of the profile.</returns>
    public async Task<string> CreateInstanceProfileWithName(
        string policyName,
        string roleName,
        string profileName,
        string ssmOnlyPolicyFile,
        List<string>? awsManagedPolicies = null)
    {
```

```
        var assumeRoleDoc = "{" +
                                "\"Version\": \"2012-10-17\"," +
                                "\"Statement\": [{" +
                                     "\"Effect\": \"Allow\"," +
                                     "\"Principal\": {" +
                                     "\"Service\": [" +
                                         "\"ec2.amazonaws.com\"" +
                                     "]" +
                                     "}," +
                                "\"Action\": \"sts:AssumeRole\"" +
                                "}]" +
                            "}";

        var policyDocument = await File.ReadAllTextAsync(ssmOnlyPolicyFile);

        var policyArn = "";

        try
        {
            var createPolicyResult = await _amazonIam.CreatePolicyAsync(
                new CreatePolicyRequest
                {
                    PolicyName = policyName,
                    PolicyDocument = policyDocument
                });
            policyArn = createPolicyResult.Policy.Arn;
        }
        catch (EntityAlreadyExistsException)
        {
            // The policy already exists, so we look it up to get the Arn.
            var policiesPaginator = _amazonIam.Paginators.ListPolicies(
                new ListPoliciesRequest()
                {
                    Scope = PolicyScopeType.Local
                });
            // Get the entire list using the paginator.
            await foreach (var policy in policiesPaginator.Policies)
            {
                if (policy.PolicyName.Equals(policyName))
                {
                    policyArn = policy.Arn;
                }
            }
```

```
            if (policyArn == null)
            {
                throw new InvalidOperationException("Policy not found");
            }
        }

        try
        {
            await _amazonIam.CreateRoleAsync(new CreateRoleRequest()
            {
                RoleName = roleName,
                AssumeRolePolicyDocument = assumeRoleDoc,
            });
            await _amazonIam.AttachRolePolicyAsync(new AttachRolePolicyRequest()
            {
                RoleName = roleName,
                PolicyArn = policyArn
            });
            if (awsManagedPolicies != null)
            {
                foreach (var awsPolicy in awsManagedPolicies)
                {
                    await _amazonIam.AttachRolePolicyAsync(new
AttachRolePolicyRequest()
                    {
                        PolicyArn = $"arn:aws:iam::aws:policy/{awsPolicy}",
                        RoleName = roleName
                    });
                }
            }
        }
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine("Role already exists.");
        }

        string profileArn = "";
        try
        {
            var profileCreateResponse = await _amazonIam.CreateInstanceProfileAsync(
                new CreateInstanceProfileRequest()
                {
                    InstanceProfileName = profileName
```

```
                });
            // Allow time for the profile to be ready.
            profileArn = profileCreateResponse.InstanceProfile.Arn;
            Thread.Sleep(10000);
            await _amazonIam.AddRoleToInstanceProfileAsync(
                new AddRoleToInstanceProfileRequest()
                {
                    InstanceProfileName = profileName,
                    RoleName = roleName
                });

        }
        catch (EntityAlreadyExistsException)
        {
            Console.WriteLine("Policy already exists.");
            var profileGetResponse = await _amazonIam.GetInstanceProfileAsync(
                new GetInstanceProfileRequest()
                {
                    InstanceProfileName = profileName
                });
            profileArn = profileGetResponse.InstanceProfile.Arn;
        }
        return profileArn;
    }

    /// <summary>
    /// Create a new key pair and save the file.
    /// </summary>
    /// <param name="newKeyPairName">The name of the new key pair.</param>
    /// <returns>Async task.</returns>
    public async Task CreateKeyPair(string newKeyPairName)
    {
        try
        {
            var keyResponse = await _amazonEc2.CreateKeyPairAsync(
                new CreateKeyPairRequest() { KeyName = newKeyPairName });
            await File.WriteAllTextAsync($"{newKeyPairName}.pem",
                keyResponse.KeyPair.KeyMaterial);
            Console.WriteLine($"Created key pair {newKeyPairName}.");
        }
        catch (AlreadyExistsException)
        {
            Console.WriteLine("Key pair already exists.");
        }
```

```
    }

    /// <summary>
    /// Delete the key pair and file by name.
    /// </summary>
    /// <param name="deleteKeyPairName">The key pair to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteKeyPairByName(string deleteKeyPairName)
    {
        try
        {
            await _amazonEc2.DeleteKeyPairAsync(
                new DeleteKeyPairRequest() { KeyName = deleteKeyPairName });
            File.Delete($"{deleteKeyPairName}.pem");
        }
        catch (FileNotFoundException)
        {
            Console.WriteLine($"Key pair {deleteKeyPairName} not found.");
        }
    }

    /// <summary>
    /// Creates an Amazon EC2 launch template to use with Amazon EC2 Auto Scaling.
    /// The launch template specifies a Bash script in its user data field that runs
after
    /// the instance is started. This script installs the Python packages and starts
a Python
    /// web server on the instance.
    /// </summary>
    /// <param name="startupScriptPath">The path to a Bash script file that is
run.</param>
    /// <param name="instancePolicyPath">The path to a permissions policy to create
and attach to the profile.</param>
    /// <returns>The template object.</returns>
    public async Task<Amazon.EC2.Model.LaunchTemplate> CreateTemplate(string
startupScriptPath, string instancePolicyPath)
    {
        await CreateKeyPair(_keyPairName);
        await CreateInstanceProfileWithName(_instancePolicyName, _instanceRoleName,
_instanceProfileName, instancePolicyPath);

        var startServerText = await File.ReadAllTextAsync(startupScriptPath);
        var plainTextBytes = System.Text.Encoding.UTF8.GetBytes(startServerText);
```

```
        var amiLatest = await _amazonSsm.GetParameterAsync(
            new GetParameterRequest() { Name = _amiParam });
        var amiId = amiLatest.Parameter.Value;
        var launchTemplateResponse = await _amazonEc2.CreateLaunchTemplateAsync(
            new CreateLaunchTemplateRequest()
            {
                LaunchTemplateName = _launchTemplateName,
                LaunchTemplateData = new RequestLaunchTemplateData()
                {
                    InstanceType = _instanceType,
                    ImageId = amiId,
                    IamInstanceProfile =
                        new
                            LaunchTemplateIamInstanceProfileSpecificationRequest()
                        {
                            Name = _instanceProfileName
                        },
                    KeyName = _keyPairName,
                    UserData = System.Convert.ToBase64String(plainTextBytes)
                }
            });
        return launchTemplateResponse.LaunchTemplate;

    }

    /// <summary>
    /// Get a list of Availability Zones in the AWS Region of the Amazon EC2 Client.
    /// </summary>
    /// <returns>A list of availability zones.</returns>
    public async Task<List<string>> DescribeAvailabilityZones()
    {
        var zoneResponse = await _amazonEc2.DescribeAvailabilityZonesAsync(
            new DescribeAvailabilityZonesRequest());
        return zoneResponse.AvailabilityZones.Select(z => z.ZoneName).ToList();
    }

    /// <summary>
    /// Create an EC2 Auto Scaling group of a specified size and name.
    /// </summary>
    /// <param name="groupSize">The size for the group.</param>
    /// <param name="groupName">The name for the group.</param>
    /// <param name="availabilityZones">The availability zones for the group.</
param>
    /// <returns>Async task.</returns>
```

```
   public async Task CreateGroupOfSize(int groupSize, string groupName,
List<string> availabilityZones)
   {
       try
       {
           await _amazonAutoScaling.CreateAutoScalingGroupAsync(
               new CreateAutoScalingGroupRequest()
               {
                   AutoScalingGroupName = groupName,
                   AvailabilityZones = availabilityZones,
                   LaunchTemplate =
                       new Amazon.AutoScaling.Model.LaunchTemplateSpecification()
                       {
                           LaunchTemplateName = _launchTemplateName,
                           Version = "$Default"
                       },
                   MaxSize = groupSize,
                   MinSize = groupSize
               });
           Console.WriteLine($"Created EC2 Auto Scaling group {groupName} with size
{groupSize}.");
       }
       catch (EntityAlreadyExistsException)
       {
           Console.WriteLine($"EC2 Auto Scaling group {groupName} already
exists.");
       }
   }

   /// <summary>
   /// Get the default VPC for the account.
   /// </summary>
   /// <returns>The default VPC object.</returns>
   public async Task<Vpc> GetDefaultVpc()
   {
       var vpcResponse = await _amazonEc2.DescribeVpcsAsync(
           new DescribeVpcsRequest()
           {
               Filters = new List<Amazon.EC2.Model.Filter>()
               {
                   new ("is-default", new List<string>() { "true" })
               }
           });
       return vpcResponse.Vpcs[0];
```

```
    }

    /// <summary>
    /// Get all the subnets for a Vpc in a set of availability zones.
    /// </summary>
    /// <param name="vpcId">The Id of the Vpc.</param>
    /// <param name="availabilityZones">The list of availability zones.</param>
    /// <returns>The collection of subnet objects.</returns>
    public async Task<List<Subnet>> GetAllVpcSubnetsForZones(string vpcId,
List<string> availabilityZones)
    {
        var subnets = new List<Subnet>();
        var subnetPaginator = _amazonEc2.Paginators.DescribeSubnets(
            new DescribeSubnetsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("vpc-id", new List<string>() { vpcId}),
                    new ("availability-zone", availabilityZones),
                    new ("default-for-az", new List<string>() { "true" })
                }
            });

        // Get the entire list using the paginator.
        await foreach (var subnet in subnetPaginator.Subnets)
        {
            subnets.Add(subnet);
        }

        return subnets;
    }

    /// <summary>
    /// Delete a launch template by name.
    /// </summary>
    /// <param name="templateName">The name of the template to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteTemplateByName(string templateName)
    {
        try
        {
            await _amazonEc2.DeleteLaunchTemplateAsync(
                new DeleteLaunchTemplateRequest()
                {
```

```
                    LaunchTemplateName = templateName
                });
        }
        catch (AmazonClientException)
        {
            Console.WriteLine($"Unable to delete template {templateName}.");
        }
    }

    /// <summary>
    /// Detaches a role from an instance profile, detaches policies from the role,
    /// and deletes all the resources.
    /// </summary>
    /// <param name="profileName">The name of the profile to delete.</param>
    /// <param name="roleName">The name of the role to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteInstanceProfile(string profileName, string roleName)
    {
        try
        {
            await _amazonIam.RemoveRoleFromInstanceProfileAsync(
                new RemoveRoleFromInstanceProfileRequest()
                {
                    InstanceProfileName = profileName,
                    RoleName = roleName
                });
            await _amazonIam.DeleteInstanceProfileAsync(
                new DeleteInstanceProfileRequest() { InstanceProfileName =
profileName });
            var attachedPolicies = await _amazonIam.ListAttachedRolePoliciesAsync(
                new ListAttachedRolePoliciesRequest() { RoleName = roleName });
            foreach (var policy in attachedPolicies.AttachedPolicies)
            {
                await _amazonIam.DetachRolePolicyAsync(
                    new DetachRolePolicyRequest()
                    {
                        RoleName = roleName,
                        PolicyArn = policy.PolicyArn
                    });
                // Delete the custom policies only.
                if (!policy.PolicyArn.StartsWith("arn:aws:iam::aws"))
                {
                    await _amazonIam.DeletePolicyAsync(
                        new Amazon.IdentityManagement.Model.DeletePolicyRequest()
```

```
                        {
                            PolicyArn = policy.PolicyArn
                        });
                }
            }

            await _amazonIam.DeleteRoleAsync(
                new DeleteRoleRequest() { RoleName = roleName });
        }
        catch (NoSuchEntityException)
        {
            Console.WriteLine($"Instance profile {profileName} does not exist.");
        }
    }

    /// <summary>
    /// Gets data about the instances in an EC2 Auto Scaling group by its group
name.
    /// </summary>
    /// <param name="group">The name of the auto scaling group.</param>
    /// <returns>A collection of instance Ids.</returns>
    public async Task<IEnumerable<string>> GetInstancesByGroupName(string group)
    {
        var instanceResponse = await
_amazonAutoScaling.DescribeAutoScalingGroupsAsync(
            new DescribeAutoScalingGroupsRequest()
            {
                AutoScalingGroupNames = new List<string>() { group }
            });
        var instanceIds = instanceResponse.AutoScalingGroups.SelectMany(
            g => g.Instances.Select(i => i.InstanceId));
        return instanceIds;
    }

    /// <summary>
    /// Get the instance profile association data for an instance.
    /// </summary>
    /// <param name="instanceId">The Id of the instance.</param>
    /// <returns>Instance profile associations data.</returns>
    public async Task<IamInstanceProfileAssociation> GetInstanceProfile(string
instanceId)
    {
        var response = await _amazonEc2.DescribeIamInstanceProfileAssociationsAsync(
            new DescribeIamInstanceProfileAssociationsRequest()
```

```
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("instance-id", new List<string>() { instanceId })
                },
            });
        return response.IamInstanceProfileAssociations[0];
    }

    /// <summary>
    /// Replace the profile associated with a running instance. After the profile is
replaced, the instance
    /// is rebooted to ensure that it uses the new profile. When the instance is
ready, Systems Manager is
    /// used to restart the Python web server.
    /// </summary>
    /// <param name="instanceId">The Id of the instance to update.</param>
    /// <param name="credsProfileName">The name of the new profile to associate with
the specified instance.</param>
    /// <param name="associationId">The Id of the existing profile association for
the instance.</param>
    /// <returns>Async task.</returns>
    public async Task ReplaceInstanceProfile(string instanceId, string
credsProfileName, string associationId)
    {
        await _amazonEc2.ReplaceIamInstanceProfileAssociationAsync(
                new ReplaceIamInstanceProfileAssociationRequest()
                {
                    AssociationId = associationId,
                    IamInstanceProfile = new IamInstanceProfileSpecification()
                    {
                        Name = credsProfileName
                    }
                });
        // Allow time before resetting.
        Thread.Sleep(25000);
        var instanceReady = false;
        var retries = 5;
        while (retries-- > 0 && !instanceReady)
        {
            await _amazonEc2.RebootInstancesAsync(
                new RebootInstancesRequest(new List<string>() { instanceId }));
            Thread.Sleep(10000);
```

```
            var instancesPaginator =
_amazonSsm.Paginators.DescribeInstanceInformation(
                new DescribeInstanceInformationRequest());
            // Get the entire list using the paginator.
            await foreach (var instance in
instancesPaginator.InstanceInformationList)
            {
                instanceReady = instance.InstanceId == instanceId;
                if (instanceReady)
                {
                    break;
                }
            }
        }
        Console.WriteLine($"Sending restart command to instance {instanceId}");
        await _amazonSsm.SendCommandAsync(
            new SendCommandRequest()
            {
                InstanceIds = new List<string>() { instanceId },
                DocumentName = "AWS-RunShellScript",
                Parameters = new Dictionary<string, List<string>>()
                {
                        {"commands", new List<string>() { "cd / && sudo python3
server.py 80" }}
                }
            });
        Console.WriteLine($"Restarted the web server on instance {instanceId}");
    }

    /// <summary>
    /// Try to terminate an instance by its Id.
    /// </summary>
    /// <param name="instanceId">The Id of the instance to terminate.</param>
    /// <returns>Async task.</returns>
    public async Task TryTerminateInstanceById(string instanceId)
    {
        var stopping = false;
        Console.WriteLine($"Stopping {instanceId}...");
        while (!stopping)
        {
            try
            {
                await _amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(
                    new TerminateInstanceInAutoScalingGroupRequest()
```

```
                    {
                        InstanceId = instanceId,
                        ShouldDecrementDesiredCapacity = false
                    });
                stopping = true;
            }
            catch (ScalingActivityInProgressException)
            {
                Console.WriteLine($"Scaling activity in progress for {instanceId}.
Waiting...");
                Thread.Sleep(10000);
            }
        }
    }

    /// <summary>
    /// Tries to delete the EC2 Auto Scaling group. If the group is in use or in
progress,
    /// waits and retries until the group is successfully deleted.
    /// </summary>
    /// <param name="groupName">The name of the group to try to delete.</param>
    /// <returns>Async task.</returns>
    public async Task TryDeleteGroupByName(string groupName)
    {
        var stopped = false;
        while (!stopped)
        {
            try
            {
                await _amazonAutoScaling.DeleteAutoScalingGroupAsync(
                    new DeleteAutoScalingGroupRequest()
                    {
                        AutoScalingGroupName = groupName
                    });
                stopped = true;
            }
            catch (Exception e)
                when ((e is ScalingActivityInProgressException)
                      || (e is Amazon.AutoScaling.Model.ResourceInUseException))
            {
                Console.WriteLine($"Some instances are still running. Waiting...");
                Thread.Sleep(10000);
            }
        }
```

```
    }

    /// <summary>
    /// Terminate instances and delete the Auto Scaling group by name.
    /// </summary>
    /// <param name="groupName">The name of the group to delete.</param>
    /// <returns>Async task.</returns>
    public async Task TerminateAndDeleteAutoScalingGroupWithName(string groupName)
    {
        var describeGroupsResponse = await
_amazonAutoScaling.DescribeAutoScalingGroupsAsync(
            new DescribeAutoScalingGroupsRequest()
            {
                AutoScalingGroupNames = new List<string>() { groupName }
            });
        if (describeGroupsResponse.AutoScalingGroups.Any())
        {
            // Update the size to 0.
            await _amazonAutoScaling.UpdateAutoScalingGroupAsync(
                new UpdateAutoScalingGroupRequest()
                {
                    AutoScalingGroupName = groupName,
                    MinSize = 0
                });
            var group = describeGroupsResponse.AutoScalingGroups[0];
            foreach (var instance in group.Instances)
            {
                await TryTerminateInstanceById(instance.InstanceId);
            }

            await TryDeleteGroupByName(groupName);
        }
        else
        {
            Console.WriteLine($"No groups found with name {groupName}.");
        }
    }


    /// <summary>
    /// Get the default security group for a specified Vpc.
    /// </summary>
    /// <param name="vpc">The Vpc to search.</param>
    /// <returns>The default security group.</returns>
```

```
    public async Task<SecurityGroup> GetDefaultSecurityGroupForVpc(Vpc vpc)
    {
        var groupResponse = await _amazonEc2.DescribeSecurityGroupsAsync(
            new DescribeSecurityGroupsRequest()
            {
                Filters = new List<Amazon.EC2.Model.Filter>()
                {
                    new ("group-name", new List<string>() { "default" }),
                    new ("vpc-id", new List<string>() { vpc.VpcId })
                }
            });
        return groupResponse.SecurityGroups[0];
    }

    /// <summary>
    /// Verify the default security group of a Vpc allows ingress from the calling
computer.
    /// This can be done by allowing ingress from this computer's IP address.
    /// In some situations, such as connecting from a corporate network, you must
instead specify
    /// a prefix list Id. You can also temporarily open the port to any IP address
while running this example.
    /// If you do, be sure to remove public access when you're done.
    /// </summary>
    /// <param name="vpc">The group to check.</param>
    /// <param name="port">The port to verify.</param>
    /// <param name="ipAddress">This computer's IP address.</param>
    /// <returns>True if the ip address is allowed on the group.</returns>
    public bool VerifyInboundPortForGroup(SecurityGroup group, int port, string
ipAddress)
    {
        var portIsOpen = false;
        foreach (var ipPermission in group.IpPermissions)
        {
            if (ipPermission.FromPort == port)
            {
                foreach (var ipRange in ipPermission.Ipv4Ranges)
                {
                    var cidr = ipRange.CidrIp;
                    if (cidr.StartsWith(ipAddress) || cidr == "0.0.0.0/0")
                    {
                        portIsOpen = true;
                    }
                }
```

```
                if (ipPermission.PrefixListIds.Any())
                {
                    portIsOpen = true;
                }

                if (!portIsOpen)
                {
                    Console.WriteLine("The inbound rule does not appear to be open
to either this computer's IP\n" +
                                      "address, to all IP addresses (0.0.0.0/0), or
to a prefix list ID.");
                }
                else
                {
                    break;
                }
            }
        }

        return portIsOpen;
    }

    /// <summary>
    /// Add an ingress rule to the specified security group that allows access on
the
    /// specified port from the specified IP address.
    /// </summary>
    /// <param name="groupId">The Id of the security group to modify.</param>
    /// <param name="port">The port to open.</param>
    /// <param name="ipAddress">The IP address to allow access.</param>
    /// <returns>Async task.</returns>
    public async Task OpenInboundPort(string groupId, int port, string ipAddress)
    {
        await _amazonEc2.AuthorizeSecurityGroupIngressAsync(
            new AuthorizeSecurityGroupIngressRequest()
            {
                GroupId = groupId,
                IpPermissions = new List<IpPermission>()
                {
                    new IpPermission()
                    {
                        FromPort = port,
                        ToPort = port,
```

```
                        IpProtocol = "tcp",
                        Ipv4Ranges = new List<IpRange>()
                        {
                            new IpRange() { CidrIp = $"{ipAddress}/32" }
                        }
                    }
                }
            });
    }

    /// <summary>
    /// Attaches an Elastic Load Balancing (ELB) target group to this EC2 Auto
 Scaling group.
    /// The
    /// </summary>
    /// <param name="autoScalingGroupName">The name of the Auto Scaling group.</
param>
    /// <param name="targetGroupArn">The Arn for the target group.</param>
    /// <returns>Async task.</returns>
    public async Task AttachLoadBalancerToGroup(string autoScalingGroupName, string
 targetGroupArn)
    {
        await _amazonAutoScaling.AttachLoadBalancerTargetGroupsAsync(
            new AttachLoadBalancerTargetGroupsRequest()
            {
                AutoScalingGroupName = autoScalingGroupName,
                TargetGroupARNs = new List<string>() { targetGroupArn }
            });
    }
}
```

Elastic Load Balancing のアクションをラップするクラスを作成します。

```
/// <summary>
/// Encapsulates Elastic Load Balancer actions.
/// </summary>
public class ElasticLoadBalancerWrapper
{
    private readonly IAmazonElasticLoadBalancingV2 _amazonElasticLoadBalancingV2;
    private string? _endpoint = null;
    private readonly string _targetGroupName = "";
```

```
    private readonly string _loadBalancerName = "";
    HttpClient _httpClient = new();

    public string TargetGroupName => _targetGroupName;
    public string LoadBalancerName => _loadBalancerName;

    /// <summary>
    /// Constructor for the Elastic Load Balancer wrapper.
    /// </summary>
    /// <param name="amazonElasticLoadBalancingV2">The injected load balancing v2
client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public ElasticLoadBalancerWrapper(
        IAmazonElasticLoadBalancingV2 amazonElasticLoadBalancingV2,
        IConfiguration configuration)
    {
        _amazonElasticLoadBalancingV2 = amazonElasticLoadBalancingV2;
        var prefix = configuration["resourcePrefix"];
        _targetGroupName = prefix + "-tg";
        _loadBalancerName = prefix + "-lb";
    }

    /// <summary>
    /// Get the HTTP Endpoint of a load balancer by its name.
    /// </summary>
    /// <param name="loadBalancerName">The name of the load balancer.</param>
    /// <returns>The HTTP endpoint.</returns>
    public async Task<string> GetEndpointForLoadBalancerByName(string
loadBalancerName)
    {
        if (_endpoint == null)
        {
            var endpointResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                    new DescribeLoadBalancersRequest()
                    {
                        Names = new List<string>() { loadBalancerName }
                    });
            _endpoint = endpointResponse.LoadBalancers[0].DNSName;
        }

        return _endpoint;
    }
```

```
    /// <summary>
    /// Return the GET response for an endpoint as text.
    /// </summary>
    /// <param name="endpoint">The endpoint for the request.</param>
    /// <returns>The request response.</returns>
    public async Task<string> GetEndPointResponse(string endpoint)
    {
        var endpointResponse = await _httpClient.GetAsync($"http://{endpoint}");
        var textResponse = await endpointResponse.Content.ReadAsStringAsync();
        return textResponse!;
    }

    /// <summary>
    /// Get the target health for a group by name.
    /// </summary>
    /// <param name="groupName">The name of the group.</param>
    /// <returns>The collection of health descriptions.</returns>
    public async Task<List<TargetHealthDescription>>
CheckTargetHealthForGroup(string groupName)
    {
        List<TargetHealthDescription> result = null!;
        try
        {
            var groupResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                    new DescribeTargetGroupsRequest()
                    {
                        Names = new List<string>() { groupName }
                    });
            var healthResponse =
                await _amazonElasticLoadBalancingV2.DescribeTargetHealthAsync(
                    new DescribeTargetHealthRequest()
                    {
                        TargetGroupArn =
groupResponse.TargetGroups[0].TargetGroupArn
                    });
            ;
            result = healthResponse.TargetHealthDescriptions;
        }
        catch (TargetGroupNotFoundException)
        {
            Console.WriteLine($"Target group {groupName} not found.");
        }
        return result;
```

```
    }

    /// <summary>
    /// Create an Elastic Load Balancing target group. The target group specifies
 how the load balancer forwards
    /// requests to instances in the group and how instance health is checked.
    ///
    /// To speed up this demo, the health check is configured with shortened times
 and lower thresholds. In production,
    /// you might want to decrease the sensitivity of your health checks to avoid
 unwanted failures.
    /// </summary>
    /// <param name="groupName">The name for the group.</param>
    /// <param name="protocol">The protocol, such as HTTP.</param>
    /// <param name="port">The port to use to forward requests, such as 80.</param>
    /// <param name="vpcId">The Id of the Vpc in which the load balancer exists.</
param>
    /// <returns>The new TargetGroup object.</returns>
    public async Task<TargetGroup> CreateTargetGroupOnVpc(string groupName,
 ProtocolEnum protocol, int port, string vpcId)
    {
        var createResponse = await
 _amazonElasticLoadBalancingV2.CreateTargetGroupAsync(
            new CreateTargetGroupRequest()
            {
                Name = groupName,
                Protocol = protocol,
                Port = port,
                HealthCheckPath = "/healthcheck",
                HealthCheckIntervalSeconds = 10,
                HealthCheckTimeoutSeconds = 5,
                HealthyThresholdCount = 2,
                UnhealthyThresholdCount = 2,
                VpcId = vpcId
            });
        var targetGroup = createResponse.TargetGroups[0];
        return targetGroup;
    }

    /// <summary>
    /// Create an Elastic Load Balancing load balancer that uses the specified
 subnets
    /// and forwards requests to the specified target group.
    /// </summary>
```

```
   /// <param name="name">The name for the new load balancer.</param>
   /// <param name="subnetIds">Subnets for the load balancer.</param>
   /// <param name="targetGroup">Target group for forwarded requests.</param>
   /// <returns>The new LoadBalancer object.</returns>
   public async Task<LoadBalancer> CreateLoadBalancerAndListener(string name,
List<string> subnetIds, TargetGroup targetGroup)
   {
       var createLbResponse = await
_amazonElasticLoadBalancingV2.CreateLoadBalancerAsync(
           new CreateLoadBalancerRequest()
           {
               Name = name,
               Subnets = subnetIds
           });
       var loadBalancerArn = createLbResponse.LoadBalancers[0].LoadBalancerArn;

       // Wait for load balancer to be available.
       var loadBalancerReady = false;
       while (!loadBalancerReady)
       {
           try
           {
               var describeResponse =
                   await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                       new DescribeLoadBalancersRequest()
                       {
                           Names = new List<string>() { name }
                       });

               var loadBalancerState =
describeResponse.LoadBalancers[0].State.Code;

               loadBalancerReady = loadBalancerState ==
LoadBalancerStateEnum.Active;
           }
           catch (LoadBalancerNotFoundException)
           {
               loadBalancerReady = false;
           }
           Thread.Sleep(10000);
       }
       // Create the listener.
       await _amazonElasticLoadBalancingV2.CreateListenerAsync(
           new CreateListenerRequest()
```

```
            {
                LoadBalancerArn = loadBalancerArn,
                Protocol = targetGroup.Protocol,
                Port = targetGroup.Port,
                DefaultActions = new List<Action>()
                {
                    new Action()
                    {
                        Type = ActionTypeEnum.Forward,
                        TargetGroupArn = targetGroup.TargetGroupArn
                    }
                }
            });
        return createLbResponse.LoadBalancers[0];
    }

    /// <summary>
    /// Verify this computer can successfully send a GET request to the
    /// load balancer endpoint.
    /// </summary>
    /// <param name="endpoint">The endpoint to check.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> VerifyLoadBalancerEndpoint(string endpoint)
    {
        var success = false;
        var retries = 3;
        while (!success && retries > 0)
        {
            try
            {
                var endpointResponse = await _httpClient.GetAsync($"http://
{endpoint}");
                Console.WriteLine($"Response: {endpointResponse.StatusCode}.");

                if (endpointResponse.IsSuccessStatusCode)
                {
                    success = true;
                }
                else
                {
                    retries = 0;
                }
            }
            catch (HttpRequestException)
```

```
            {
                Console.WriteLine("Connection error, retrying...");
                retries--;
                Thread.Sleep(10000);
            }
        }

        return success;
    }

    /// <summary>
    /// Delete a load balancer by its specified name.
    /// </summary>
    /// <param name="name">The name of the load balancer to delete.</param>
    /// <returns>Async task.</returns>
    public async Task DeleteLoadBalancerByName(string name)
    {
        try
        {
            var describeLoadBalancerResponse =
                await _amazonElasticLoadBalancingV2.DescribeLoadBalancersAsync(
                    new DescribeLoadBalancersRequest()
                    {
                        Names = new List<string>() { name }
                    });
            var lbArn =
describeLoadBalancerResponse.LoadBalancers[0].LoadBalancerArn;
            await _amazonElasticLoadBalancingV2.DeleteLoadBalancerAsync(
                new DeleteLoadBalancerRequest()
                {
                    LoadBalancerArn = lbArn
                }
            );
        }
        catch (LoadBalancerNotFoundException)
        {
            Console.WriteLine($"Load balancer {name} not found.");
        }
    }

    /// <summary>
    /// Delete a TargetGroup by its specified name.
    /// </summary>
    /// <param name="groupName">Name of the group to delete.</param>
```

```
    /// <returns>Async task.</returns>
    public async Task DeleteTargetGroupByName(string groupName)
    {
        var done = false;
        while (!done)
        {
            try
            {
                var groupResponse =
                    await _amazonElasticLoadBalancingV2.DescribeTargetGroupsAsync(
                        new DescribeTargetGroupsRequest()
                        {
                            Names = new List<string>() { groupName }
                        });

                var targetArn = groupResponse.TargetGroups[0].TargetGroupArn;
                await _amazonElasticLoadBalancingV2.DeleteTargetGroupAsync(
                    new DeleteTargetGroupRequest() { TargetGroupArn = targetArn });
                Console.WriteLine($"Deleted load balancing target group
 {groupName}.");
                done = true;
            }
            catch (TargetGroupNotFoundException)
            {
                Console.WriteLine(
                    $"Target group {groupName} not found, could not delete.");
                done = true;
            }
            catch (ResourceInUseException)
            {
                Console.WriteLine("Target group not yet released, waiting...");
                Thread.Sleep(10000);
            }
        }
    }
}
```

DynamoDB を使用してレコメンデーションサービスをシミュレートするクラスを作成します。

```
/// <summary>
/// Encapsulates a DynamoDB table to use as a service that recommends books, movies,
 and songs.
```

```
/// </summary>
public class Recommendations
{
    private readonly IAmazonDynamoDB _amazonDynamoDb;
    private readonly DynamoDBContext _context;
    private readonly string _tableName;

    public string TableName => _tableName;

    /// <summary>
    /// Constructor for the Recommendations service.
    /// </summary>
    /// <param name="amazonDynamoDb">The injected DynamoDb client.</param>
    /// <param name="configuration">The injected configuration.</param>
    public Recommendations(IAmazonDynamoDB amazonDynamoDb, IConfiguration
 configuration)
    {
        _amazonDynamoDb = amazonDynamoDb;
        _context = new DynamoDBContext(_amazonDynamoDb);
        _tableName = configuration["databaseName"]!;
    }

    /// <summary>
    /// Create the DynamoDb table with a specified name.
    /// </summary>
    /// <param name="tableName">The name for the table.</param>
    /// <returns>True when ready.</returns>
    public async Task<bool> CreateDatabaseWithName(string tableName)
    {
        try
        {
            Console.Write($"Creating table {tableName}...");
            var createRequest = new CreateTableRequest()
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                    {
                        new AttributeDefinition()
                        {
                            AttributeName = "MediaType",
                            AttributeType = ScalarAttributeType.S
                        },
                        new AttributeDefinition()
                        {
```

```
                        AttributeName = "ItemId",
                        AttributeType = ScalarAttributeType.N
                    }
                },
            KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement()
                    {
                        AttributeName = "MediaType",
                        KeyType = KeyType.HASH
                    },
                    new KeySchemaElement()
                    {
                        AttributeName = "ItemId",
                        KeyType = KeyType.RANGE
                    }
                },
            ProvisionedThroughput = new ProvisionedThroughput()
            {
                ReadCapacityUnits = 5,
                WriteCapacityUnits = 5
            }
        };
        await _amazonDynamoDb.CreateTableAsync(createRequest);

        // Wait until the table is ACTIVE and then report success.
        Console.Write("\nWaiting for table to become active...");

        var request = new DescribeTableRequest
        {
            TableName = tableName
        };

        TableStatus status;
        do
        {
            Thread.Sleep(2000);

            var describeTableResponse = await
_amazonDynamoDb.DescribeTableAsync(request);
            status = describeTableResponse.Table.TableStatus;

            Console.Write(".");
        }
```

```
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException)
        {
            Console.WriteLine($"Table {tableName} already exists.");
            return false;
        }
    }

    /// <summary>
    /// Populate the database table with data from a specified path.
    /// </summary>
    /// <param name="databaseTableName">The name of the table.</param>
    /// <param name="recommendationsPath">The path of the recommendations data.</
param>
    /// <returns>Async task.</returns>
    public async Task PopulateDatabase(string databaseTableName, string
 recommendationsPath)
    {
        var recommendationsText = await File.ReadAllTextAsync(recommendationsPath);
        var records =
            JsonSerializer.Deserialize<RecommendationModel[]>(recommendationsText);
        var batchWrite = _context.CreateBatchWrite<RecommendationModel>();

        foreach (var record in records!)
        {
            batchWrite.AddPutItem(record);
        }

        await batchWrite.ExecuteAsync();
    }

    /// <summary>
    /// Delete the recommendation table by name.
    /// </summary>
    /// <param name="tableName">The name of the recommendation table.</param>
    /// <returns>Async task.</returns>
    public async Task DestroyDatabaseByName(string tableName)
    {
        try
        {
            await _amazonDynamoDb.DeleteTableAsync(
```

```
                new DeleteTableRequest() { TableName = tableName });
            Console.WriteLine($"Table {tableName} was deleted.");
        }
        catch (ResourceNotFoundException)
        {
            Console.WriteLine($"Table {tableName} not found");
        }
    }
}
```

Systems Manager のアクションをラップするクラスを作成します。

```
/// <summary>
/// Encapsulates Systems Manager parameter operations. This example uses these
 parameters
/// to drive the demonstration of resilient architecture, such as failure of a
 dependency or
/// how the service responds to a health check.
/// </summary>
public class SmParameterWrapper
{
    private readonly IAmazonSimpleSystemsManagement _amazonSimpleSystemsManagement;

    private readonly string _tableParameter = "doc-example-resilient-architecture-
table";
    private readonly string _failureResponseParameter = "doc-example-resilient-
architecture-failure-response";
    private readonly string _healthCheckParameter = "doc-example-resilient-
architecture-health-check";
    private readonly string _tableName = "";

    public string TableParameter => _tableParameter;
    public string TableName => _tableName;
    public string HealthCheckParameter => _healthCheckParameter;
    public string FailureResponseParameter => _failureResponseParameter;

    /// <summary>
    /// Constructor for the SmParameterWrapper.
    /// </summary>
    /// <param name="amazonSimpleSystemsManagement">The injected Simple Systems
 Management client.</param>
    /// <param name="configuration">The injected configuration.</param>
```

```
    public SmParameterWrapper(IAmazonSimpleSystemsManagement
 amazonSimpleSystemsManagement, IConfiguration configuration)
    {
        _amazonSimpleSystemsManagement = amazonSimpleSystemsManagement;
        _tableName = configuration["databaseName"]!;
    }

    /// <summary>
    /// Reset the Systems Manager parameters to starting values for the demo.
    /// </summary>
    /// <returns>Async task.</returns>
    public async Task Reset()
    {
        await this.PutParameterByName(_tableParameter, _tableName);
        await this.PutParameterByName(_failureResponseParameter, "none");
        await this.PutParameterByName(_healthCheckParameter, "shallow");
    }

    /// <summary>
    /// Set the value of a named Systems Manager parameter.
    /// </summary>
    /// <param name="name">The name of the parameter.</param>
    /// <param name="value">The value to set.</param>
    /// <returns>Async task.</returns>
    public async Task PutParameterByName(string name, string value)
    {
        await _amazonSimpleSystemsManagement.PutParameterAsync(
            new PutParameterRequest() { Name = name, Value = value, Overwrite =
 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。
  - AttachLoadBalancerTargetGroups
  - CreateAutoScalingGroup
  - CreateInstanceProfile
  - CreateLaunchTemplate
  - CreateListener
  - CreateLoadBalancer

- CreateTargetGroup
- DeleteAutoScalingGroup
- DeleteInstanceProfile
- DeleteLaunchTemplate
- DeleteLoadBalancer
- DeleteTargetGroup
- DescribeAutoScalingGroups
- DescribeAvailabilityZones
- DescribeIamInstanceProfileAssociations
- DescribeInstances
- DescribeLoadBalancers
- DescribeSubnets
- DescribeTargetGroups
- DescribeTargetHealth
- DescribeVpcs
- RebootInstances
- ReplaceIamInstanceProfileAssociation
- TerminateInstanceInAutoScalingGroup
- UpdateAutoScalingGroup

グループを作成しユーザーを追加します。

次のコードサンプルは、以下の操作方法を示しています。

- グループを作成し、そのグループに Amazon S3 のフルアクセス許可を付与します。
- Amazon S3 にアクセス許可のない新しいユーザーを作成します。
- ユーザーをグループに追加し、そのユーザーが Amazon S3 のアクセス許可を持っていることを確認してから、リソースをクリーンアップします。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
global using Amazon.IdentityManagement;
global using Amazon.S3;
global using Amazon.SecurityToken;
global using IAMActions;
global using IamScenariosCommon;
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;


namespace IAMActions;

public class IAMWrapper
{
    private readonly IAmazonIdentityManagementService _IAMService;

    /// <summary>
    /// Constructor for the IAMWrapper class.
    /// </summary>
    /// <param name="IAMService">An IAM client object.</param>
    public IAMWrapper(IAmazonIdentityManagementService IAMService)
    {
        _IAMService = IAMService;
    }

    /// <summary>
    /// Add an existing IAM user to an existing IAM group.
    /// </summary>
    /// <param name="userName">The username of the user to add.</param>
    /// <param name="groupName">The name of the group to add the user to.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AddUserToGroupAsync(string userName, string groupName)
```

```
    {
        var response = await _IAMService.AddUserToGroupAsync(new
AddUserToGroupRequest
        {
            GroupName = groupName,
            UserName = userName,
        });

        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Attach an IAM policy to a role.
    /// </summary>
    /// <param name="policyArn">The policy to attach.</param>
    /// <param name="roleName">The role that the policy will be attached to.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AttachRolePolicyAsync(string policyArn, string roleName)
    {
        var response = await _IAMService.AttachRolePolicyAsync(new
AttachRolePolicyRequest
        {
            PolicyArn = policyArn,
            RoleName = roleName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Create an IAM access key for a user.
    /// </summary>
    /// <param name="userName">The username for which to create the IAM access
    /// key.</param>
    /// <returns>The AccessKey.</returns>
    public async Task<AccessKey> CreateAccessKeyAsync(string userName)
    {
        var response = await _IAMService.CreateAccessKeyAsync(new
CreateAccessKeyRequest
        {
            UserName = userName,
        });
```

```
        return response.AccessKey;

    }


    /// <summary>
    /// Create an IAM group.
    /// </summary>
    /// <param name="groupName">The name to give the IAM group.</param>
    /// <returns>The IAM group that was created.</returns>
    public async Task<Group> CreateGroupAsync(string groupName)
    {
        var response = await _IAMService.CreateGroupAsync(new CreateGroupRequest
{ GroupName = groupName });
        return response.Group;
    }


    /// <summary>
    /// Create an IAM policy.
    /// </summary>
    /// <param name="policyName">The name to give the new IAM policy.</param>
    /// <param name="policyDocument">The policy document for the new policy.</param>
    /// <returns>The new IAM policy object.</returns>
    public async Task<ManagedPolicy> CreatePolicyAsync(string policyName, string
policyDocument)
    {
        var response = await _IAMService.CreatePolicyAsync(new CreatePolicyRequest
        {
            PolicyDocument = policyDocument,
            PolicyName = policyName,
        });

        return response.Policy;
    }


    /// <summary>
    /// Create a new IAM role.
    /// </summary>
    /// <param name="roleName">The name of the IAM role.</param>
    /// <param name="rolePolicyDocument">The name of the IAM policy document
    /// for the new role.</param>
```

```
    /// <returns>The Amazon Resource Name (ARN) of the role.</returns>
    public async Task<string> CreateRoleAsync(string roleName, string
 rolePolicyDocument)
    {
        var request = new CreateRoleRequest
        {
            RoleName = roleName,
            AssumeRolePolicyDocument = rolePolicyDocument,
        };

        var response = await _IAMService.CreateRoleAsync(request);
        return response.Role.Arn;
    }


    /// <summary>
    /// Create an IAM service-linked role.
    /// </summary>
    /// <param name="serviceName">The name of the AWS Service.</param>
    /// <param name="description">A description of the IAM service-linked role.</
param>
    /// <returns>The IAM role that was created.</returns>
    public async Task<Role> CreateServiceLinkedRoleAsync(string serviceName, string
 description)
    {
        var request = new CreateServiceLinkedRoleRequest
        {
            AWSServiceName = serviceName,
            Description = description
        };

        var response = await _IAMService.CreateServiceLinkedRoleAsync(request);
        return response.Role;
    }


    /// <summary>
    /// Create an IAM user.
    /// </summary>
    /// <param name="userName">The username for the new IAM user.</param>
    /// <returns>The IAM user that was created.</returns>
    public async Task<User> CreateUserAsync(string userName)
    {
```

```
        var response = await _IAMService.CreateUserAsync(new CreateUserRequest
{ UserName = userName });
        return response.User;
    }


    /// <summary>
    /// Delete an IAM user's access key.
    /// </summary>
    /// <param name="accessKeyId">The Id for the IAM access key.</param>
    /// <param name="userName">The username of the user that owns the IAM
    /// access key.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteAccessKeyAsync(string accessKeyId, string
userName)
    {
        var response = await _IAMService.DeleteAccessKeyAsync(new
DeleteAccessKeyRequest
        {
            AccessKeyId = accessKeyId,
            UserName = userName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an IAM group.
    /// </summary>
    /// <param name="groupName">The name of the IAM group to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteGroupAsync(string groupName)
    {
        var response = await _IAMService.DeleteGroupAsync(new DeleteGroupRequest
{ GroupName = groupName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an IAM policy associated with an IAM group.
    /// </summary>
    /// <param name="groupName">The name of the IAM group associated with the
```

```
    /// policy.</param>
    /// <param name="policyName">The name of the policy to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteGroupPolicyAsync(string groupName, string
policyName)
    {
        var request = new DeleteGroupPolicyRequest()
        {
            GroupName = groupName,
            PolicyName = policyName,
        };

        var response = await _IAMService.DeleteGroupPolicyAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an IAM policy.
    /// </summary>
    /// <param name="policyArn">The Amazon Resource Name (ARN) of the policy to
    /// delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeletePolicyAsync(string policyArn)
    {
        var response = await _IAMService.DeletePolicyAsync(new DeletePolicyRequest
{ PolicyArn = policyArn });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an IAM role.
    /// </summary>
    /// <param name="roleName">The name of the IAM role to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteRoleAsync(string roleName)
    {
        var response = await _IAMService.DeleteRoleAsync(new DeleteRoleRequest
{ RoleName = roleName });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

```
   /// <summary>
   /// Delete an IAM role policy.
   /// </summary>
   /// <param name="roleName">The name of the IAM role.</param>
   /// <param name="policyName">The name of the IAM role policy to delete.</param>
   /// <returns>A Boolean value indicating the success of the action.</returns>
   public async Task<bool> DeleteRolePolicyAsync(string roleName, string
policyName)
   {
       var response = await _IAMService.DeleteRolePolicyAsync(new
DeleteRolePolicyRequest
       {
           PolicyName = policyName,
           RoleName = roleName,
       });

       return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
   }


   /// <summary>
   /// Delete an IAM user.
   /// </summary>
   /// <param name="userName">The username of the IAM user to delete.</param>
   /// <returns>A Boolean value indicating the success of the action.</returns>
   public async Task<bool> DeleteUserAsync(string userName)
   {
       var response = await _IAMService.DeleteUserAsync(new DeleteUserRequest
{ UserName = userName });

       return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
   }


   /// <summary>
   /// Delete an IAM user policy.
   /// </summary>
   /// <param name="policyName">The name of the IAM policy to delete.</param>
   /// <param name="userName">The username of the IAM user.</param>
   /// <returns>A Boolean value indicating the success of the action.</returns>
   public async Task<bool> DeleteUserPolicyAsync(string policyName, string
userName)
   {
```

```
        var response = await _IAMService.DeleteUserPolicyAsync(new
 DeleteUserPolicyRequest { PolicyName = policyName, UserName = userName });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Detach an IAM policy from an IAM role.
    /// </summary>
    /// <param name="policyArn">The Amazon Resource Name (ARN) of the IAM policy.</
param>
    /// <param name="roleName">The name of the IAM role.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DetachRolePolicyAsync(string policyArn, string roleName)
    {
        var response = await _IAMService.DetachRolePolicyAsync(new
 DetachRolePolicyRequest
        {
            PolicyArn = policyArn,
            RoleName = roleName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Gets the IAM password policy for an AWS account.
    /// </summary>
    /// <returns>The PasswordPolicy for the AWS account.</returns>
    public async Task<PasswordPolicy> GetAccountPasswordPolicyAsync()
    {
        var response = await _IAMService.GetAccountPasswordPolicyAsync(new
 GetAccountPasswordPolicyRequest());
        return response.PasswordPolicy;
    }


    /// <summary>
    /// Get information about an IAM policy.
    /// </summary>
    /// <param name="policyArn">The IAM policy to retrieve information for.</param>
    /// <returns>The IAM policy.</returns>
```

```
    public async Task<ManagedPolicy> GetPolicyAsync(string policyArn)
    {

        var response = await _IAMService.GetPolicyAsync(new GetPolicyRequest
{ PolicyArn = policyArn });
        return response.Policy;
    }


    /// <summary>
    /// Get information about an IAM role.
    /// </summary>
    /// <param name="roleName">The name of the IAM role to retrieve information
    /// for.</param>
    /// <returns>The IAM role that was retrieved.</returns>
    public async Task<Role> GetRoleAsync(string roleName)
    {
        var response = await _IAMService.GetRoleAsync(new GetRoleRequest
        {
            RoleName = roleName,
        });

        return response.Role;
    }


    /// <summary>
    /// Get information about an IAM user.
    /// </summary>
    /// <param name="userName">The username of the user.</param>
    /// <returns>An IAM user object.</returns>
    public async Task<User> GetUserAsync(string userName)
    {
        var response = await _IAMService.GetUserAsync(new GetUserRequest { UserName
= userName });
        return response.User;
    }


    /// <summary>
    /// List the IAM role policies that are attached to an IAM role.
    /// </summary>
    /// <param name="roleName">The IAM role to list IAM policies for.</param>
    /// <returns>A list of the IAM policies attached to the IAM role.</returns>
```

```
   public async Task<List<AttachedPolicyType>> ListAttachedRolePoliciesAsync(string
roleName)
   {
       var attachedPolicies = new List<AttachedPolicyType>();
       var attachedRolePoliciesPaginator =
_IAMService.Paginators.ListAttachedRolePolicies(new ListAttachedRolePoliciesRequest
{ RoleName = roleName });

       await foreach (var response in attachedRolePoliciesPaginator.Responses)
       {
           attachedPolicies.AddRange(response.AttachedPolicies);
       }

       return attachedPolicies;
   }


   /// <summary>
   /// List IAM groups.
   /// </summary>
   /// <returns>A list of IAM groups.</returns>
   public async Task<List<Group>> ListGroupsAsync()
   {
       var groupsPaginator = _IAMService.Paginators.ListGroups(new
ListGroupsRequest());
       var groups = new List<Group>();

       await foreach (var response in groupsPaginator.Responses)
       {
           groups.AddRange(response.Groups);
       }

       return groups;
   }


   /// <summary>
   /// List IAM policies.
   /// </summary>
   /// <returns>A list of the IAM policies.</returns>
   public async Task<List<ManagedPolicy>> ListPoliciesAsync()
   {
       var listPoliciesPaginator = _IAMService.Paginators.ListPolicies(new
ListPoliciesRequest());
```

```
        var policies = new List<ManagedPolicy>();

        await foreach (var response in listPoliciesPaginator.Responses)
        {
            policies.AddRange(response.Policies);
        }

        return policies;
    }


    /// <summary>
    /// List IAM role policies.
    /// </summary>
    /// <param name="roleName">The IAM role for which to list IAM policies.</param>
    /// <returns>A list of IAM policy names.</returns>
    public async Task<List<string>> ListRolePoliciesAsync(string roleName)
    {
        var listRolePoliciesPaginator = _IAMService.Paginators.ListRolePolicies(new
ListRolePoliciesRequest { RoleName = roleName });
        var policyNames = new List<string>();

        await foreach (var response in listRolePoliciesPaginator.Responses)
        {
            policyNames.AddRange(response.PolicyNames);
        }

        return policyNames;
    }


    /// <summary>
    /// List IAM roles.
    /// </summary>
    /// <returns>A list of IAM roles.</returns>
    public async Task<List<Role>> ListRolesAsync()
    {
        var listRolesPaginator = _IAMService.Paginators.ListRoles(new
ListRolesRequest());
        var roles = new List<Role>();

        await foreach (var response in listRolesPaginator.Responses)
        {
            roles.AddRange(response.Roles);
```

```
        }

        return roles;
    }


    /// <summary>
    /// List SAML authentication providers.
    /// </summary>
    /// <returns>A list of SAML providers.</returns>
    public async Task<List<SAMLProviderListEntry>> ListSAMLProvidersAsync()
    {
        var response = await _IAMService.ListSAMLProvidersAsync(new
 ListSAMLProvidersRequest());
        return response.SAMLProviderList;
    }


    /// <summary>
    /// List IAM users.
    /// </summary>
    /// <returns>A list of IAM users.</returns>
    public async Task<List<User>> ListUsersAsync()
    {
        var listUsersPaginator = _IAMService.Paginators.ListUsers(new
 ListUsersRequest());
        var users = new List<User>();

        await foreach (var response in listUsersPaginator.Responses)
        {
            users.AddRange(response.Users);
        }

        return users;
    }


    /// <summary>
    /// Remove a user from an IAM group.
    /// </summary>
    /// <param name="userName">The username of the user to remove.</param>
    /// <param name="groupName">The name of the IAM group to remove the user from.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
```

```
    public async Task<bool> RemoveUserFromGroupAsync(string userName, string
 groupName)
    {
        // Remove the user from the group.
        var removeUserRequest = new RemoveUserFromGroupRequest()
        {
            UserName = userName,
            GroupName = groupName,
        };

        var response = await
 _IAMService.RemoveUserFromGroupAsync(removeUserRequest);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Add or update an inline policy document that is embedded in an IAM group.
    /// </summary>
    /// <param name="groupName">The name of the IAM group.</param>
    /// <param name="policyName">The name of the IAM policy.</param>
    /// <param name="policyDocument">The policy document defining the IAM policy.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> PutGroupPolicyAsync(string groupName, string policyName,
 string policyDocument)
    {
        var request = new PutGroupPolicyRequest
        {
            GroupName = groupName,
            PolicyName = policyName,
            PolicyDocument = policyDocument
        };

        var response = await _IAMService.PutGroupPolicyAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Update the inline policy document embedded in a role.
    /// </summary>
    /// <param name="policyName">The name of the policy to embed.</param>
    /// <param name="roleName">The name of the role to update.</param>
```

```
    /// <param name="policyDocument">The policy document that defines the role.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> PutRolePolicyAsync(string policyName, string roleName,
 string policyDocument)
    {
        var request = new PutRolePolicyRequest
        {
            PolicyName = policyName,
            RoleName = roleName,
            PolicyDocument = policyDocument
        };

        var response = await _IAMService.PutRolePolicyAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Add or update an inline policy document that is embedded in an IAM user.
    /// </summary>
    /// <param name="userName">The name of the IAM user.</param>
    /// <param name="policyName">The name of the IAM policy.</param>
    /// <param name="policyDocument">The policy document defining the IAM policy.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> PutUserPolicyAsync(string userName, string policyName,
 string policyDocument)
    {
        var request = new PutUserPolicyRequest
        {
            UserName = userName,
            PolicyName = policyName,
            PolicyDocument = policyDocument
        };

        var response = await _IAMService.PutUserPolicyAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Wait for a new access key to be ready to use.
    /// </summary>
    /// <param name="accessKeyId">The Id of the access key.</param>
```

```
    /// <returns>A boolean value indicating the success of the action.</returns>
    public async Task<bool> WaitUntilAccessKeyIsReady(string accessKeyId)
    {
        var keyReady = false;

        do
        {
            try
            {
                var response = await _IAMService.GetAccessKeyLastUsedAsync(
                    new GetAccessKeyLastUsedRequest { AccessKeyId = accessKeyId });
                if (response.UserName is not null)
                {
                    keyReady = true;
                }
            }
            catch (NoSuchEntityException)
            {
                keyReady = false;
            }
        } while (!keyReady);

        return keyReady;
    }
}


using Microsoft.Extensions.Configuration;

namespace IAMGroups;

public class IAMGroups
{
    private static ILogger logger = null!;

    // Represents JSON code for AWS full access policy for Amazon Simple
    // Storage Service (Amazon S3).
    private const string S3FullAccessPolicyDocument = "{" +
        " \"Statement\" : [{" +
            " \"Action\" : [\"s3:*\"]," +
            " \"Effect\" : \"Allow\"," +
            " \"Resource\" : \"*\"" +
        "}]" +
    "}";
```

```
   static async Task Main(string[] args)
   {
       // Set up dependency injection for the AWS 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<IAmazonIdentityManagementService>()
           .AddTransient<IAMWrapper>()
           .AddTransient<UIWrapper>()
           )
           .Build();

       logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
           .CreateLogger<IAMGroups>();

       IConfiguration 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 groupUserName = configuration["GroupUserName"];
       var groupName = configuration["GroupName"];
       var groupPolicyName = configuration["GroupPolicyName"];
       var groupBucketName = configuration["GroupBucketName"];

       var wrapper = host.Services.GetRequiredService<IAMWrapper>();
       var uiWrapper = host.Services.GetRequiredService<UIWrapper>();

       uiWrapper.DisplayGroupsOverview();
       uiWrapper.PressEnter();

       // Create an IAM group.
       uiWrapper.DisplayTitle("Create IAM group");
       Console.WriteLine("Let's begin by creating a new IAM group.");
       var group = await wrapper.CreateGroupAsync(groupName);
```

```
        // Add an inline IAM policy to the group.
        uiWrapper.DisplayTitle("Add policy to group");
        Console.WriteLine("Add an inline policy to the group that allows members to
have full access to");
        Console.WriteLine("Amazon Simple Storage Service (Amazon S3) buckets.");

        await wrapper.PutGroupPolicyAsync(group.GroupName, groupPolicyName,
S3FullAccessPolicyDocument);

        uiWrapper.PressEnter();

        // Now create a new user.
        uiWrapper.DisplayTitle("Create an IAM user");
        Console.WriteLine("Now let's create a new IAM user.");
        var groupUser = await wrapper.CreateUserAsync(groupUserName);

        // Add the new user to the group.
        uiWrapper.DisplayTitle("Add the user to the group");
        Console.WriteLine("Adding the user to the group, which will give the user
the same permissions as the group.");
        await wrapper.AddUserToGroupAsync(groupUser.UserName, group.GroupName);

        Console.WriteLine($"User, {groupUser.UserName}, has been added to the group,
{group.GroupName}.");
        uiWrapper.PressEnter();

        Console.WriteLine("Now that we have created a user, and added the user to
the group, let's create an IAM access key.");

        // Create access and secret keys for the user.
        var accessKey = await wrapper.CreateAccessKeyAsync(groupUserName);
        Console.WriteLine("Key created.");
        uiWrapper.WaitABit(15, "Waiting for the access key to be ready for use.");

        uiWrapper.DisplayTitle("List buckets");
        Console.WriteLine("To prove that the user has access to Amazon S3, list the
S3 buckets for the account.");

        var s3Client = new AmazonS3Client(accessKey.AccessKeyId,
accessKey.SecretAccessKey);
        var stsClient = new AmazonSecurityTokenServiceClient(accessKey.AccessKeyId,
accessKey.SecretAccessKey);

        var s3Wrapper = new S3Wrapper(s3Client, stsClient);
```

```
        var buckets = await s3Wrapper.ListMyBucketsAsync();

        if (buckets is not null)
        {
            buckets.ForEach(bucket =>
            {
                Console.WriteLine($"{bucket.BucketName}\tcreated on:
{bucket.CreationDate}");
            });
        }

        // Show that the user also has write access to Amazon S3 by creating
        // a new bucket.
        uiWrapper.DisplayTitle("Create a bucket");
        Console.WriteLine("Since group members have full access to Amazon S3, let's
create a bucket.");
        var success = await s3Wrapper.PutBucketAsync(groupBucketName);

        if (success)
        {
            Console.WriteLine($"Successfully created the bucket:
{groupBucketName}.");
        }

        uiWrapper.PressEnter();

        Console.WriteLine("Let's list the user's S3 buckets again to show the new
bucket.");

        buckets = await s3Wrapper.ListMyBucketsAsync();

        if (buckets is not null)
        {
            buckets.ForEach(bucket =>
            {
                Console.WriteLine($"{bucket.BucketName}\tcreated on:
{bucket.CreationDate}");
            });
        }

        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Clean up resources");
```

```
        Console.WriteLine("First delete the bucket we created.");
        await s3Wrapper.DeleteBucketAsync(groupBucketName);

        Console.WriteLine($"Now remove the user, {groupUserName}, from the group,
 {groupName}.");
        await wrapper.RemoveUserFromGroupAsync(groupUserName, groupName);

        Console.WriteLine("Delete the user's access key.");
        await wrapper.DeleteAccessKeyAsync(accessKey.AccessKeyId, groupUserName);

        // Now we can safely delete the user.
        Console.WriteLine("Now we can delete the user.");
        await wrapper.DeleteUserAsync(groupUserName);

        uiWrapper.PressEnter();

        Console.WriteLine("Now we will delete the IAM policy attached to the
 group.");
        await wrapper.DeleteGroupPolicyAsync(groupName, groupPolicyName);

        Console.WriteLine("Now we delete the IAM group.");
        await wrapper.DeleteGroupAsync(groupName);

        uiWrapper.PressEnter();

        Console.WriteLine("The IAM groups demo has completed.");

        uiWrapper.PressEnter();
    }
}


namespace IamScenariosCommon;

using System.Net;

/// <summary>
/// A class to perform Amazon Simple Storage Service (Amazon S3) actions for
/// the IAM Basics scenario.
/// </summary>
public class S3Wrapper
{
    private IAmazonS3 _s3Service;
    private IAmazonSecurityTokenService _stsService;
```

```
    /// <summary>
    /// Constructor for the S3Wrapper class.
    /// </summary>
    /// <param name="s3Service">An Amazon S3 client object.</param>
    /// <param name="stsService">An AWS Security Token Service (AWS STS)
    /// client object.</param>
    public S3Wrapper(IAmazonS3 s3Service, IAmazonSecurityTokenService stsService)
    {
        _s3Service = s3Service;
        _stsService = stsService;
    }

    /// <summary>
    /// Assumes an AWS Identity and Access Management (IAM) role that allows
    /// Amazon S3 access for the current session.
    /// </summary>
    /// <param name="roleSession">A string representing the current session.</param>
    /// <param name="roleToAssume">The name of the IAM role to assume.</param>
    /// <returns>Credentials for the newly assumed IAM role.</returns>
    public async Task<Credentials> AssumeS3RoleAsync(string roleSession, string
roleToAssume)
    {
        // Create the request to use with the AssumeRoleAsync call.
        var request = new AssumeRoleRequest()
        {
            RoleSessionName = roleSession,
            RoleArn = roleToAssume,
        };

        var response = await _stsService.AssumeRoleAsync(request);

        return response.Credentials;
    }


    /// <summary>
    /// Delete an S3 bucket.
    /// </summary>
    /// <param name="bucketName">Name of the S3 bucket to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteBucketAsync(string bucketName)
    {
```

```
        var result = await _s3Service.DeleteBucketAsync(new DeleteBucketRequest
{ BucketName = bucketName });
        return result.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// List the buckets that are owned by the user's account.
    /// </summary>
    /// <returns>Async Task.</returns>
    public async Task<List<S3Bucket>?> ListMyBucketsAsync()
    {
        try
        {
            // Get the list of buckets accessible by the new user.
            var response = await _s3Service.ListBucketsAsync();

            return response.Buckets;
        }
        catch (AmazonS3Exception ex)
        {
            // Something else went wrong. Display the error message.
            Console.WriteLine($"Error: {ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Create a new S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name for the new bucket.</param>
    /// <returns>A Boolean value indicating whether the action completed
    /// successfully.</returns>
    public async Task<bool> PutBucketAsync(string bucketName)
    {
        var response = await _s3Service.PutBucketAsync(new PutBucketRequest
{ BucketName = bucketName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Update the client objects with new client objects. This is available
    /// because the scenario uses the methods of this class without and then
    /// with the proper permissions to list S3 buckets.
    /// </summary>
```

```
    /// <param name="s3Service">The Amazon S3 client object.</param>
    /// <param name="stsService">The AWS STS client object.</param>
    public void UpdateClients(IAmazonS3 s3Service, IAmazonSecurityTokenService
 stsService)
    {
        _s3Service = s3Service;
        _stsService = stsService;
    }
}


namespace IamScenariosCommon;

public class UIWrapper
{
    public readonly string SepBar = new('-', Console.WindowWidth);

    /// <summary>
    /// Show information about the IAM Groups scenario.
    /// </summary>
    public void DisplayGroupsOverview()
    {
        Console.Clear();

        DisplayTitle("Welcome to the IAM Groups Demo");
        Console.WriteLine("This example application does the following:");
        Console.WriteLine("\t1. Creates an Amazon Identity and Access Management
 (IAM) group.");
        Console.WriteLine("\t2. Adds an IAM policy to the IAM group giving it full
 access to Amazon S3.");
        Console.WriteLine("\t3. Creates a new IAM user.");
        Console.WriteLine("\t4. Creates an IAM access key for the user.");
        Console.WriteLine("\t5. Adds the user to the IAM group.");
        Console.WriteLine("\t6. Lists the buckets on the account.");
        Console.WriteLine("\t7. Proves that the user has full Amazon S3 access by
 creating a bucket.");
        Console.WriteLine("\t8. List the buckets again to show the new bucket.");
        Console.WriteLine("\t9. Cleans up all the resources created.");
    }

    /// <summary>
    /// Show information about the IAM Basics scenario.
    /// </summary>
    public void DisplayBasicsOverview()
```

```
    {
        Console.Clear();

        DisplayTitle("Welcome to IAM Basics");
        Console.WriteLine("This example application does the following:");
        Console.WriteLine("\t1. Creates a user with no permissions.");
        Console.WriteLine("\t2. Creates a role and policy that grant
s3:ListAllMyBuckets permission.");
        Console.WriteLine("\t3. Grants the user permission to assume the role.");
        Console.WriteLine("\t4. Creates an S3 client object as the user and tries to
list buckets (this will fail).");
        Console.WriteLine("\t5. Gets temporary credentials by assuming the role.");
        Console.WriteLine("\t6. Creates a new S3 client object with the temporary
credentials and lists the buckets (this will succeed).");
        Console.WriteLine("\t7. Deletes all the resources.");
    }

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

- 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。
  - AddUserToGroup
  - AttachRolePolicy
  - CreateAccessKey
  - CreateGroup
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessKey
  - DeleteGroup

- DeleteGroupPolicy
- DeleteUser
- PutGroupPolicy
- RemoveUserFromGroup

ユーザーを作成してロールを引き受ける

次のコードサンプルは、ユーザーを作成してロールを割り当てる方法を示しています。

> **Warning**
>
> セキュリティリスクを避けるため、専用ソフトウェアの開発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 Identity Center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。

- 権限のないユーザーを作成します。
- 指定したアカウントに Amazon S3 バケットへのアクセス権限を付与するロールを作成します。
- ユーザーにロールを引き受けさせるポリシーを追加します。
- ロールを引き受け、一時的な認証情報を使用して S3 バケットを一覧表示しリソースをクリーンアップします。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
global using Amazon.IdentityManagement;
global using Amazon.S3;
global using Amazon.SecurityToken;
global using IAMActions;
global using IamScenariosCommon;
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;


namespace IAMActions;

public class IAMWrapper
{
    private readonly IAmazonIdentityManagementService _IAMService;

    /// <summary>
    /// Constructor for the IAMWrapper class.
    /// </summary>
    /// <param name="IAMService">An IAM client object.</param>
    public IAMWrapper(IAmazonIdentityManagementService IAMService)
    {
        _IAMService = IAMService;
    }

    /// <summary>
    /// Add an existing IAM user to an existing IAM group.
    /// </summary>
    /// <param name="userName">The username of the user to add.</param>
    /// <param name="groupName">The name of the group to add the user to.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AddUserToGroupAsync(string userName, string groupName)
    {
        var response = await _IAMService.AddUserToGroupAsync(new
 AddUserToGroupRequest
        {
            GroupName = groupName,
            UserName = userName,
        });

        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Attach an IAM policy to a role.
    /// </summary>
    /// <param name="policyArn">The policy to attach.</param>
    /// <param name="roleName">The role that the policy will be attached to.</param>
```

```
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AttachRolePolicyAsync(string policyArn, string roleName)
    {
        var response = await _IAMService.AttachRolePolicyAsync(new
AttachRolePolicyRequest
        {
            PolicyArn = policyArn,
            RoleName = roleName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Create an IAM access key for a user.
    /// </summary>
    /// <param name="userName">The username for which to create the IAM access
    /// key.</param>
    /// <returns>The AccessKey.</returns>
    public async Task<AccessKey> CreateAccessKeyAsync(string userName)
    {
        var response = await _IAMService.CreateAccessKeyAsync(new
CreateAccessKeyRequest
        {
            UserName = userName,
        });

        return response.AccessKey;

    }


    /// <summary>
    /// Create an IAM group.
    /// </summary>
    /// <param name="groupName">The name to give the IAM group.</param>
    /// <returns>The IAM group that was created.</returns>
    public async Task<Group> CreateGroupAsync(string groupName)
    {
        var response = await _IAMService.CreateGroupAsync(new CreateGroupRequest
{ GroupName = groupName });
        return response.Group;
    }
```

```
   /// <summary>
   /// Create an IAM policy.
   /// </summary>
   /// <param name="policyName">The name to give the new IAM policy.</param>
   /// <param name="policyDocument">The policy document for the new policy.</param>
   /// <returns>The new IAM policy object.</returns>
   public async Task<ManagedPolicy> CreatePolicyAsync(string policyName, string
policyDocument)
   {
       var response = await _IAMService.CreatePolicyAsync(new CreatePolicyRequest
       {
           PolicyDocument = policyDocument,
           PolicyName = policyName,
       });

       return response.Policy;
   }


   /// <summary>
   /// Create a new IAM role.
   /// </summary>
   /// <param name="roleName">The name of the IAM role.</param>
   /// <param name="rolePolicyDocument">The name of the IAM policy document
   /// for the new role.</param>
   /// <returns>The Amazon Resource Name (ARN) of the role.</returns>
   public async Task<string> CreateRoleAsync(string roleName, string
rolePolicyDocument)
   {
       var request = new CreateRoleRequest
       {
           RoleName = roleName,
           AssumeRolePolicyDocument = rolePolicyDocument,
       };

       var response = await _IAMService.CreateRoleAsync(request);
       return response.Role.Arn;
   }


   /// <summary>
   /// Create an IAM service-linked role.
```

```
    /// </summary>
    /// <param name="serviceName">The name of the AWS Service.</param>
    /// <param name="description">A description of the IAM service-linked role.</
param>
    /// <returns>The IAM role that was created.</returns>
    public async Task<Role> CreateServiceLinkedRoleAsync(string serviceName, string
 description)
    {
        var request = new CreateServiceLinkedRoleRequest
        {
            AWSServiceName = serviceName,
            Description = description
        };

        var response = await _IAMService.CreateServiceLinkedRoleAsync(request);
        return response.Role;
    }


    /// <summary>
    /// Create an IAM user.
    /// </summary>
    /// <param name="userName">The username for the new IAM user.</param>
    /// <returns>The IAM user that was created.</returns>
    public async Task<User> CreateUserAsync(string userName)
    {
        var response = await _IAMService.CreateUserAsync(new CreateUserRequest
 { UserName = userName });
        return response.User;
    }


    /// <summary>
    /// Delete an IAM user's access key.
    /// </summary>
    /// <param name="accessKeyId">The Id for the IAM access key.</param>
    /// <param name="userName">The username of the user that owns the IAM
    /// access key.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteAccessKeyAsync(string accessKeyId, string
 userName)
    {
        var response = await _IAMService.DeleteAccessKeyAsync(new
 DeleteAccessKeyRequest
```

```
        {
            AccessKeyId = accessKeyId,
            UserName = userName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an IAM group.
    /// </summary>
    /// <param name="groupName">The name of the IAM group to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteGroupAsync(string groupName)
    {
        var response = await _IAMService.DeleteGroupAsync(new DeleteGroupRequest
{ GroupName = groupName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an IAM policy associated with an IAM group.
    /// </summary>
    /// <param name="groupName">The name of the IAM group associated with the
    /// policy.</param>
    /// <param name="policyName">The name of the policy to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteGroupPolicyAsync(string groupName, string
policyName)
    {
        var request = new DeleteGroupPolicyRequest()
        {
            GroupName = groupName,
            PolicyName = policyName,
        };

        var response = await _IAMService.DeleteGroupPolicyAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
```

```
    /// Delete an IAM policy.
    /// </summary>
    /// <param name="policyArn">The Amazon Resource Name (ARN) of the policy to
    /// delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeletePolicyAsync(string policyArn)
    {
        var response = await _IAMService.DeletePolicyAsync(new DeletePolicyRequest
{ PolicyArn = policyArn });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an IAM role.
    /// </summary>
    /// <param name="roleName">The name of the IAM role to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteRoleAsync(string roleName)
    {
        var response = await _IAMService.DeleteRoleAsync(new DeleteRoleRequest
{ RoleName = roleName });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an IAM role policy.
    /// </summary>
    /// <param name="roleName">The name of the IAM role.</param>
    /// <param name="policyName">The name of the IAM role policy to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteRolePolicyAsync(string roleName, string
policyName)
    {
        var response = await _IAMService.DeleteRolePolicyAsync(new
DeleteRolePolicyRequest
        {
            PolicyName = policyName,
            RoleName = roleName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

```
    /// <summary>
    /// Delete an IAM user.
    /// </summary>
    /// <param name="userName">The username of the IAM user to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteUserAsync(string userName)
    {
        var response = await _IAMService.DeleteUserAsync(new DeleteUserRequest
 { UserName = userName });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an IAM user policy.
    /// </summary>
    /// <param name="policyName">The name of the IAM policy to delete.</param>
    /// <param name="userName">The username of the IAM user.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteUserPolicyAsync(string policyName, string
 userName)
    {
        var response = await _IAMService.DeleteUserPolicyAsync(new
 DeleteUserPolicyRequest { PolicyName = policyName, UserName = userName });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Detach an IAM policy from an IAM role.
    /// </summary>
    /// <param name="policyArn">The Amazon Resource Name (ARN) of the IAM policy.</
param>
    /// <param name="roleName">The name of the IAM role.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DetachRolePolicyAsync(string policyArn, string roleName)
    {
        var response = await _IAMService.DetachRolePolicyAsync(new
 DetachRolePolicyRequest
        {
```

```
            PolicyArn = policyArn,
            RoleName = roleName,
        });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Gets the IAM password policy for an AWS account.
    /// </summary>
    /// <returns>The PasswordPolicy for the AWS account.</returns>
    public async Task<PasswordPolicy> GetAccountPasswordPolicyAsync()
    {
        var response = await _IAMService.GetAccountPasswordPolicyAsync(new
GetAccountPasswordPolicyRequest());
        return response.PasswordPolicy;
    }


    /// <summary>
    /// Get information about an IAM policy.
    /// </summary>
    /// <param name="policyArn">The IAM policy to retrieve information for.</param>
    /// <returns>The IAM policy.</returns>
    public async Task<ManagedPolicy> GetPolicyAsync(string policyArn)
    {

        var response = await _IAMService.GetPolicyAsync(new GetPolicyRequest
{ PolicyArn = policyArn });
        return response.Policy;
    }


    /// <summary>
    /// Get information about an IAM role.
    /// </summary>
    /// <param name="roleName">The name of the IAM role to retrieve information
    /// for.</param>
    /// <returns>The IAM role that was retrieved.</returns>
    public async Task<Role> GetRoleAsync(string roleName)
    {
        var response = await _IAMService.GetRoleAsync(new GetRoleRequest
        {
```

```
            RoleName = roleName,
        });

        return response.Role;
    }


    /// <summary>
    /// Get information about an IAM user.
    /// </summary>
    /// <param name="userName">The username of the user.</param>
    /// <returns>An IAM user object.</returns>
    public async Task<User> GetUserAsync(string userName)
    {
        var response = await _IAMService.GetUserAsync(new GetUserRequest { UserName
= userName });
        return response.User;
    }


    /// <summary>
    /// List the IAM role policies that are attached to an IAM role.
    /// </summary>
    /// <param name="roleName">The IAM role to list IAM policies for.</param>
    /// <returns>A list of the IAM policies attached to the IAM role.</returns>
    public async Task<List<AttachedPolicyType>> ListAttachedRolePoliciesAsync(string
roleName)
    {
        var attachedPolicies = new List<AttachedPolicyType>();
        var attachedRolePoliciesPaginator =
_IAMService.Paginators.ListAttachedRolePolicies(new ListAttachedRolePoliciesRequest
{ RoleName = roleName });

        await foreach (var response in attachedRolePoliciesPaginator.Responses)
        {
            attachedPolicies.AddRange(response.AttachedPolicies);
        }

        return attachedPolicies;
    }


    /// <summary>
    /// List IAM groups.
```

```
    /// </summary>
    /// <returns>A list of IAM groups.</returns>
    public async Task<List<Group>> ListGroupsAsync()
    {
        var groupsPaginator = _IAMService.Paginators.ListGroups(new
ListGroupsRequest());
        var groups = new List<Group>();

        await foreach (var response in groupsPaginator.Responses)
        {
            groups.AddRange(response.Groups);
        }

        return groups;
    }


    /// <summary>
    /// List IAM policies.
    /// </summary>
    /// <returns>A list of the IAM policies.</returns>
    public async Task<List<ManagedPolicy>> ListPoliciesAsync()
    {
        var listPoliciesPaginator = _IAMService.Paginators.ListPolicies(new
ListPoliciesRequest());
        var policies = new List<ManagedPolicy>();

        await foreach (var response in listPoliciesPaginator.Responses)
        {
            policies.AddRange(response.Policies);
        }

        return policies;
    }


    /// <summary>
    /// List IAM role policies.
    /// </summary>
    /// <param name="roleName">The IAM role for which to list IAM policies.</param>
    /// <returns>A list of IAM policy names.</returns>
    public async Task<List<string>> ListRolePoliciesAsync(string roleName)
    {
```

```
        var listRolePoliciesPaginator = _IAMService.Paginators.ListRolePolicies(new
ListRolePoliciesRequest { RoleName = roleName });
        var policyNames = new List<string>();

        await foreach (var response in listRolePoliciesPaginator.Responses)
        {
            policyNames.AddRange(response.PolicyNames);
        }

        return policyNames;
    }


    /// <summary>
    /// List IAM roles.
    /// </summary>
    /// <returns>A list of IAM roles.</returns>
    public async Task<List<Role>> ListRolesAsync()
    {
        var listRolesPaginator = _IAMService.Paginators.ListRoles(new
ListRolesRequest());
        var roles = new List<Role>();

        await foreach (var response in listRolesPaginator.Responses)
        {
            roles.AddRange(response.Roles);
        }

        return roles;
    }


    /// <summary>
    /// List SAML authentication providers.
    /// </summary>
    /// <returns>A list of SAML providers.</returns>
    public async Task<List<SAMLProviderListEntry>> ListSAMLProvidersAsync()
    {
        var response = await _IAMService.ListSAMLProvidersAsync(new
ListSAMLProvidersRequest());
        return response.SAMLProviderList;
    }
```

```
    /// <summary>
    /// List IAM users.
    /// </summary>
    /// <returns>A list of IAM users.</returns>
    public async Task<List<User>> ListUsersAsync()
    {
        var listUsersPaginator = _IAMService.Paginators.ListUsers(new
 ListUsersRequest());
        var users = new List<User>();

        await foreach (var response in listUsersPaginator.Responses)
        {
            users.AddRange(response.Users);
        }

        return users;
    }


    /// <summary>
    /// Remove a user from an IAM group.
    /// </summary>
    /// <param name="userName">The username of the user to remove.</param>
    /// <param name="groupName">The name of the IAM group to remove the user from.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> RemoveUserFromGroupAsync(string userName, string
 groupName)
    {
        // Remove the user from the group.
        var removeUserRequest = new RemoveUserFromGroupRequest()
        {
            UserName = userName,
            GroupName = groupName,
        };

        var response = await
 _IAMService.RemoveUserFromGroupAsync(removeUserRequest);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Add or update an inline policy document that is embedded in an IAM group.
```

```
    /// </summary>
    /// <param name="groupName">The name of the IAM group.</param>
    /// <param name="policyName">The name of the IAM policy.</param>
    /// <param name="policyDocument">The policy document defining the IAM policy.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> PutGroupPolicyAsync(string groupName, string policyName,
 string policyDocument)
    {
        var request = new PutGroupPolicyRequest
        {
            GroupName = groupName,
            PolicyName = policyName,
            PolicyDocument = policyDocument
        };

        var response = await _IAMService.PutGroupPolicyAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Update the inline policy document embedded in a role.
    /// </summary>
    /// <param name="policyName">The name of the policy to embed.</param>
    /// <param name="roleName">The name of the role to update.</param>
    /// <param name="policyDocument">The policy document that defines the role.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> PutRolePolicyAsync(string policyName, string roleName,
 string policyDocument)
    {
        var request = new PutRolePolicyRequest
        {
            PolicyName = policyName,
            RoleName = roleName,
            PolicyDocument = policyDocument
        };

        var response = await _IAMService.PutRolePolicyAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

```
    /// <summary>
    /// Add or update an inline policy document that is embedded in an IAM user.
    /// </summary>
    /// <param name="userName">The name of the IAM user.</param>
    /// <param name="policyName">The name of the IAM policy.</param>
    /// <param name="policyDocument">The policy document defining the IAM policy.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> PutUserPolicyAsync(string userName, string policyName,
 string policyDocument)
    {
        var request = new PutUserPolicyRequest
        {
            UserName = userName,
            PolicyName = policyName,
            PolicyDocument = policyDocument
        };

        var response = await _IAMService.PutUserPolicyAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Wait for a new access key to be ready to use.
    /// </summary>
    /// <param name="accessKeyId">The Id of the access key.</param>
    /// <returns>A boolean value indicating the success of the action.</returns>
    public async Task<bool> WaitUntilAccessKeyIsReady(string accessKeyId)
    {
        var keyReady = false;

        do
        {
            try
            {
                var response = await _IAMService.GetAccessKeyLastUsedAsync(
                    new GetAccessKeyLastUsedRequest { AccessKeyId = accessKeyId });
                if (response.UserName is not null)
                {
                    keyReady = true;
                }
            }
            catch (NoSuchEntityException)
            {
```

```
                keyReady = false;
            }
        } while (!keyReady);

        return keyReady;
    }
}



using Microsoft.Extensions.Configuration;

namespace IAMBasics;

public class IAMBasics
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for the AWS 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<IAmazonIdentityManagementService>()
            .AddTransient<IAMWrapper>()
            .AddTransient<UIWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<IAMBasics>();


        IConfiguration 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();
```

```
        // Values needed for user, role, and policies.
        string userName = configuration["UserName"]!;
        string s3PolicyName = configuration["S3PolicyName"]!;
        string roleName = configuration["RoleName"]!;


        var iamWrapper = host.Services.GetRequiredService<IAMWrapper>();
        var uiWrapper = host.Services.GetRequiredService<UIWrapper>();

        uiWrapper.DisplayBasicsOverview();
        uiWrapper.PressEnter();

        // First create a user. By default, the new user has
        // no permissions.
        uiWrapper.DisplayTitle("Create User");
        Console.WriteLine($"Creating a new user with user name: {userName}.");
        var user = await iamWrapper.CreateUserAsync(userName);
        var userArn = user.Arn;

        Console.WriteLine($"Successfully created user: {userName} with ARN:
{userArn}.");
        uiWrapper.WaitABit(15, "Now let's wait for the user to be ready for use.");

        // Define a role policy document that allows the new user
        // to assume the role.
        string assumeRolePolicyDocument = "{" +
          "\"Version\": \"2012-10-17\"," +
          "\"Statement\": [{" +
              "\"Effect\": \"Allow\"," +
              "\"Principal\": {" +
              $" \"AWS\": \"{userArn}\"" +
              "}," +
              "\"Action\": \"sts:AssumeRole\"" +
          "}]" +
        "}";

        // Permissions to list all buckets.
        string policyDocument = "{" +
            "\"Version\": \"2012-10-17\"," +
            " \"Statement\" : [{" +
                " \"Action\" : [\"s3:ListAllMyBuckets\"]," +
                " \"Effect\" : \"Allow\"," +
                " \"Resource\" : \"*\"" +
```

```
            "}]" +
        "}";

        // Create an AccessKey for the user.
        uiWrapper.DisplayTitle("Create access key");
        Console.WriteLine("Now let's create an access key for the new user.");
        var accessKey = await iamWrapper.CreateAccessKeyAsync(userName);

        var accessKeyId = accessKey.AccessKeyId;
        var secretAccessKey = accessKey.SecretAccessKey;

        Console.WriteLine($"We have created the access key with Access key id:
{accessKeyId}.");

        Console.WriteLine("Now let's wait until the IAM access key is ready to
use.");
        var keyReady = await iamWrapper.WaitUntilAccessKeyIsReady(accessKeyId);

        // Now try listing the Amazon Simple Storage Service (Amazon S3)
        // buckets. This should fail at this point because the user doesn't
        // have permissions to perform this task.
        uiWrapper.DisplayTitle("Try to display Amazon S3 buckets");
        Console.WriteLine("Now let's try to display a list of the user's Amazon S3
buckets.");
        var s3Client1 = new AmazonS3Client(accessKeyId, secretAccessKey);
        var stsClient1 = new AmazonSecurityTokenServiceClient(accessKeyId,
secretAccessKey);

        var s3Wrapper = new S3Wrapper(s3Client1, stsClient1);
        var buckets = await s3Wrapper.ListMyBucketsAsync();

        Console.WriteLine(buckets is null
            ? "As expected, the call to list the buckets has returned a null list."
            : "Something went wrong. This shouldn't have worked.");

        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Create IAM role");
        Console.WriteLine($"Creating the role: {roleName}");

        // Creating an IAM role to allow listing the S3 buckets. A role name
        // is not case sensitive and must be unique to the account for which it
        // is created.
```

```
        var roleArn = await iamWrapper.CreateRoleAsync(roleName,
 assumeRolePolicyDocument);

        uiWrapper.PressEnter();

        // Create a policy with permissions to list S3 buckets.
        uiWrapper.DisplayTitle("Create IAM policy");
        Console.WriteLine($"Creating the policy: {s3PolicyName}");
        Console.WriteLine("with permissions to list the Amazon S3 buckets for the
 account.");
        var policy = await iamWrapper.CreatePolicyAsync(s3PolicyName,
 policyDocument);

        // Wait 15 seconds for the IAM policy to be available.
        uiWrapper.WaitABit(15, "Waiting for the policy to be available.");

        // Attach the policy to the role you created earlier.
        uiWrapper.DisplayTitle("Attach new IAM policy");
        Console.WriteLine("Now let's attach the policy to the role.");
        await iamWrapper.AttachRolePolicyAsync(policy.Arn, roleName);

        // Wait 15 seconds for the role to be updated.
        Console.WriteLine();
        uiWrapper.WaitABit(15, "Waiting for the policy to be attached.");

        // Use the AWS Security Token Service (AWS STS) to have the user
        // assume the role we created.
        var stsClient2 = new AmazonSecurityTokenServiceClient(accessKeyId,
 secretAccessKey);

        // Wait for the new credentials to become valid.
        uiWrapper.WaitABit(10, "Waiting for the credentials to be valid.");

        var assumedRoleCredentials = await s3Wrapper.AssumeS3RoleAsync("temporary-
session", roleArn);

        // Try again to list the buckets using the client created with
        // the new user's credentials. This time, it should work.
        var s3Client2 = new AmazonS3Client(assumedRoleCredentials);

        s3Wrapper.UpdateClients(s3Client2, stsClient2);

        buckets = await s3Wrapper.ListMyBucketsAsync();
```

```
        uiWrapper.DisplayTitle("List Amazon S3 buckets");
        Console.WriteLine("This time we should have buckets to list.");
        if (buckets is not null)
        {
            buckets.ForEach(bucket =>
            {
                Console.WriteLine($"{bucket.BucketName} created:
 {bucket.CreationDate}");
            });
        }

        uiWrapper.PressEnter();

        // Now clean up all the resources used in the example.
        uiWrapper.DisplayTitle("Clean up resources");
        Console.WriteLine("Thank you for watching. The IAM Basics demo is
 complete.");
        Console.WriteLine("Please wait while we clean up the resources we
 created.");

        await iamWrapper.DetachRolePolicyAsync(policy.Arn, roleName);

        await iamWrapper.DeletePolicyAsync(policy.Arn);

        await iamWrapper.DeleteRoleAsync(roleName);

        await iamWrapper.DeleteAccessKeyAsync(accessKeyId, userName);

        await iamWrapper.DeleteUserAsync(userName);

        uiWrapper.PressEnter();

        Console.WriteLine("All done cleaning up our resources. Thank you for your
 patience.");
    }
}


namespace IamScenariosCommon;

using System.Net;

/// <summary>
/// A class to perform Amazon Simple Storage Service (Amazon S3) actions for
```

```
/// the IAM Basics scenario.
/// </summary>
public class S3Wrapper
{
    private IAmazonS3 _s3Service;
    private IAmazonSecurityTokenService _stsService;

    /// <summary>
    /// Constructor for the S3Wrapper class.
    /// </summary>
    /// <param name="s3Service">An Amazon S3 client object.</param>
    /// <param name="stsService">An AWS Security Token Service (AWS STS)
    /// client object.</param>
    public S3Wrapper(IAmazonS3 s3Service, IAmazonSecurityTokenService stsService)
    {
        _s3Service = s3Service;
        _stsService = stsService;
    }

    /// <summary>
    /// Assumes an AWS Identity and Access Management (IAM) role that allows
    /// Amazon S3 access for the current session.
    /// </summary>
    /// <param name="roleSession">A string representing the current session.</param>
    /// <param name="roleToAssume">The name of the IAM role to assume.</param>
    /// <returns>Credentials for the newly assumed IAM role.</returns>
    public async Task<Credentials> AssumeS3RoleAsync(string roleSession, string
 roleToAssume)
    {
        // Create the request to use with the AssumeRoleAsync call.
        var request = new AssumeRoleRequest()
        {
            RoleSessionName = roleSession,
            RoleArn = roleToAssume,
        };

        var response = await _stsService.AssumeRoleAsync(request);

        return response.Credentials;
    }


    /// <summary>
    /// Delete an S3 bucket.
```

```
    /// </summary>
    /// <param name="bucketName">Name of the S3 bucket to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteBucketAsync(string bucketName)
    {
        var result = await _s3Service.DeleteBucketAsync(new DeleteBucketRequest
{ BucketName = bucketName });
        return result.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// List the buckets that are owned by the user's account.
    /// </summary>
    /// <returns>Async Task.</returns>
    public async Task<List<S3Bucket>?> ListMyBucketsAsync()
    {
        try
        {
            // Get the list of buckets accessible by the new user.
            var response = await _s3Service.ListBucketsAsync();

            return response.Buckets;
        }
        catch (AmazonS3Exception ex)
        {
            // Something else went wrong. Display the error message.
            Console.WriteLine($"Error: {ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Create a new S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name for the new bucket.</param>
    /// <returns>A Boolean value indicating whether the action completed
    /// successfully.</returns>
    public async Task<bool> PutBucketAsync(string bucketName)
    {
        var response = await _s3Service.PutBucketAsync(new PutBucketRequest
{ BucketName = bucketName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

```
    /// <summary>
    /// Update the client objects with new client objects. This is available
    /// because the scenario uses the methods of this class without and then
    /// with the proper permissions to list S3 buckets.
    /// </summary>
    /// <param name="s3Service">The Amazon S3 client object.</param>
    /// <param name="stsService">The AWS STS client object.</param>
    public void UpdateClients(IAmazonS3 s3Service, IAmazonSecurityTokenService
 stsService)
    {
        _s3Service = s3Service;
        _stsService = stsService;
    }
}


namespace IamScenariosCommon;

public class UIWrapper
{
    public readonly string SepBar = new('-', Console.WindowWidth);

    /// <summary>
    /// Show information about the IAM Groups scenario.
    /// </summary>
    public void DisplayGroupsOverview()
    {
        Console.Clear();

        DisplayTitle("Welcome to the IAM Groups Demo");
        Console.WriteLine("This example application does the following:");
        Console.WriteLine("\t1. Creates an Amazon Identity and Access Management
 (IAM) group.");
        Console.WriteLine("\t2. Adds an IAM policy to the IAM group giving it full
 access to Amazon S3.");
        Console.WriteLine("\t3. Creates a new IAM user.");
        Console.WriteLine("\t4. Creates an IAM access key for the user.");
        Console.WriteLine("\t5. Adds the user to the IAM group.");
        Console.WriteLine("\t6. Lists the buckets on the account.");
        Console.WriteLine("\t7. Proves that the user has full Amazon S3 access by
 creating a bucket.");
        Console.WriteLine("\t8. List the buckets again to show the new bucket.");
        Console.WriteLine("\t9. Cleans up all the resources created.");
    }
```

```
    /// <summary>
    /// Show information about the IAM Basics scenario.
    /// </summary>
    public void DisplayBasicsOverview()
    {
        Console.Clear();

        DisplayTitle("Welcome to IAM Basics");
        Console.WriteLine("This example application does the following:");
        Console.WriteLine("\t1. Creates a user with no permissions.");
        Console.WriteLine("\t2. Creates a role and policy that grant
s3:ListAllMyBuckets permission.");
        Console.WriteLine("\t3. Grants the user permission to assume the role.");
        Console.WriteLine("\t4. Creates an S3 client object as the user and tries to
list buckets (this will fail).");
        Console.WriteLine("\t5. Gets temporary credentials by assuming the role.");
        Console.WriteLine("\t6. Creates a new S3 client object with the temporary
credentials and lists the buckets (this will succeed).");
        Console.WriteLine("\t7. Deletes all the resources.");
    }

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

- 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。
  - AttachRolePolicy
  - CreateAccessKey
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessKey

- DeletePolicy
- DeleteRole
- DeleteUser
- DeleteUserPolicy
- DetachRolePolicy
- PutUserPolicy

を使用した Amazon Keyspaces の例 AWS SDK for .NET

次のコード例は、Amazon Keyspaces AWS SDK for .NET で 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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。このリンクには GitHub、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が記載されています。

開始方法

Hello Amazon Keyspaces

次のコード例は、Amazon Keyspaces の使用を開始する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
namespace KeyspacesActions;
```

```
public class HelloKeyspaces
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon Keyspaces (for Apache Cassandra).
        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<IAmazonKeyspaces>()
                .AddTransient<KeyspacesWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<HelloKeyspaces>();

        var keyspacesClient = host.Services.GetRequiredService<IAmazonKeyspaces>();
        var keyspacesWrapper = new KeyspacesWrapper(keyspacesClient);

        Console.WriteLine("Hello, Amazon Keyspaces! Let's list your keyspaces:");
        await keyspacesWrapper.ListKeyspaces();
    }
}
```

- API の詳細については、「 API リファレンスListKeyspaces」の「」を参照してください。 AWS SDK for .NET

トピック

- アクション
- シナリオ

アクション

`CreateKeyspace`

次の例は、CreateKeyspace を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
/// Create a new keyspace.
/// </summary>
/// <param name="keyspaceName">The name for the new keyspace.</param>
/// <returns>The Amazon Resource Name (ARN) of the new keyspace.</returns>
public async Task<string> CreateKeyspace(string keyspaceName)
{
    var response =
        await _amazonKeyspaces.CreateKeyspaceAsync(
            new CreateKeyspaceRequest { KeyspaceName = keyspaceName });
    return response.ResourceArn;
}
```

- API の詳細については、「 API リファレンスCreateKeyspace」の「」を参照してください。 AWS SDK for .NET

`CreateTable`

次の例は、CreateTable を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Create a new Amazon Keyspaces table.
    /// </summary>
    /// <param name="keyspaceName">The keyspace where the table will be created.</
param>
    /// <param name="schema">The schema for the new table.</param>
    /// <param name="tableName">The name of the new table.</param>
    /// <returns>The Amazon Resource Name (ARN) of the new table.</returns>
    public async Task<string> CreateTable(string keyspaceName, SchemaDefinition
 schema, string tableName)
    {
        var request = new CreateTableRequest
        {
            KeyspaceName = keyspaceName,
            SchemaDefinition = schema,
            TableName = tableName,
            PointInTimeRecovery = new PointInTimeRecovery { Status =
 PointInTimeRecoveryStatus.ENABLED }
        };

        var response = await _amazonKeyspaces.CreateTableAsync(request);
        return response.ResourceArn;
    }
```

- API の詳細については、「 API リファレンスCreateTable」の「」を参照してください。 AWS SDK for .NET

`DeleteKeyspace`

次の例は、DeleteKeyspace を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
/// Delete an existing keyspace.
/// </summary>
/// <param name="keyspaceName"></param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteKeyspace(string keyspaceName)
{
    var response = await _amazonKeyspaces.DeleteKeyspaceAsync(
        new DeleteKeyspaceRequest { KeyspaceName = keyspaceName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- API の詳細については、「 API リファレンスDeleteKeyspace」の「」を参照してください。AWS SDK for .NET

`DeleteTable`

次の例は、`DeleteTable` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
/// Delete an Amazon Keyspaces table.
/// </summary>
```

```
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteTable(string keyspaceName, string tableName)
    {
        var response = await _amazonKeyspaces.DeleteTableAsync(
            new DeleteTableRequest { KeyspaceName = keyspaceName, TableName =
tableName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスDeleteTable」の「」を参照してください。 AWS SDK for .NET

`GetKeyspace`

次の例は、`GetKeyspace` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get data about a keyspace.
    /// </summary>
    /// <param name="keyspaceName">The name of the keyspace.</param>
    /// <returns>The Amazon Resource Name (ARN) of the keyspace.</returns>
    public async Task<string> GetKeyspace(string keyspaceName)
    {
        var response = await _amazonKeyspaces.GetKeyspaceAsync(
            new GetKeyspaceRequest { KeyspaceName = keyspaceName });
        return response.ResourceArn;
    }
```

- API の詳細については、「 API リファレンスGetKeyspace」の「」を参照してください。 AWS SDK for .NET

`GetTable`

次の例は、GetTable を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get information about an Amazon Keyspaces table.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the Amazon Keyspaces table.</param>
    /// <returns>The response containing data about the table.</returns>
    public async Task<GetTableResponse> GetTable(string keyspaceName, string
tableName)
    {
        var response = await _amazonKeyspaces.GetTableAsync(
            new GetTableRequest { KeyspaceName = keyspaceName, TableName =
tableName });
        return response;
    }
```

- API の詳細については、「 API リファレンスGetTable」の「」を参照してください。 AWS SDK for .NET

`ListKeyspaces`

次の例は、ListKeyspaces を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Lists all keyspaces for the account.
    /// </summary>
    /// <returns>Async task.</returns>
    public async Task ListKeyspaces()
    {
        var paginator = _amazonKeyspaces.Paginators.ListKeyspaces(new
ListKeyspacesRequest());

        Console.WriteLine("{0, -30}\t{1}", "Keyspace name", "Keyspace ARN");
        Console.WriteLine(new string('-', Console.WindowWidth));
        await foreach (var keyspace in paginator.Keyspaces)
        {

Console.WriteLine($"{keyspace.KeyspaceName,-30}\t{keyspace.ResourceArn}");
        }
    }
```

- API の詳細については、「 API リファレンスListKeyspaces」の「」を参照してください。AWS SDK for .NET

`ListTables`

次の例は、`ListTables` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Lists the Amazon Keyspaces tables in a keyspace.
    /// </summary>
    /// <param name="keyspaceName">The name of the keyspace.</param>
    /// <returns>A list of TableSummary objects.</returns>
    public async Task<List<TableSummary>> ListTables(string keyspaceName)
    {
        var response = await _amazonKeyspaces.ListTablesAsync(new ListTablesRequest
{ KeyspaceName = keyspaceName });
        response.Tables.ForEach(table =>
        {

Console.WriteLine($"{table.KeyspaceName}\t{table.TableName}\t{table.ResourceArn}");
        });

        return response.Tables;
    }
```

- API の詳細については、「 API リファレンスListTables」の「」を参照してください。 AWS SDK for .NET

`RestoreTable`

次の例は、RestoreTable を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Restores the specified table to the specified point in time.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table to restore.</param>
    /// <param name="timestamp">The time to which the table will be restored.</
param>
    /// <returns>The Amazon Resource Name (ARN) of the restored table.</returns>
    public async Task<string> RestoreTable(string keyspaceName, string tableName,
 string restoredTableName, DateTime timestamp)
    {
        var request = new RestoreTableRequest
        {
            RestoreTimestamp = timestamp,
            SourceKeyspaceName = keyspaceName,
            SourceTableName = tableName,
            TargetKeyspaceName = keyspaceName,
            TargetTableName = restoredTableName
        };

        var response = await _amazonKeyspaces.RestoreTableAsync(request);
        return response.RestoredTableARN;
    }
```

- API の詳細については、「 API リファレンスRestoreTable」の「」を参照してください。AWS SDK for .NET

`UpdateTable`

次の例は、UpdateTable を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
/// Updates the movie table to add a boolean column named watched.
/// </summary>
/// <param name="keyspaceName">The keyspace containing the table.</param>
/// <param name="tableName">The name of the table to change.</param>
/// <returns>The Amazon Resource Name (ARN) of the updated table.</returns>
public async Task<string> UpdateTable(string keyspaceName, string tableName)
{
    var newColumn = new ColumnDefinition { Name = "watched", Type = "boolean" };
    var request = new UpdateTableRequest
    {
        KeyspaceName = keyspaceName,
        TableName = tableName,
        AddColumns = new List<ColumnDefinition> { newColumn }
    };
    var response = await _amazonKeyspaces.UpdateTableAsync(request);
    return response.ResourceArn;
}
```

- API の詳細については、「 API リファレンスUpdateTable」の「」を参照してください。 AWS SDK for .NET

シナリオ

キースペースとテーブルの使用を開始する

次のコードサンプルは、以下の操作方法を示しています。

- キースペースとテーブルを作成する。テーブルスキーマは映画データを保持し、 point-in-time 復旧が有効になっています。
- SIGv4 認証による安全な TLS 接続を使用してキースペースに接続します。

- テーブルに対してクエリを実行します。ムービーデータを追加、取得、更新します。
- テーブルを更新する。視聴したムービーを追跡する列を追加します。
- テーブルを以前の状態に戻し、リソースをクリーンアップします。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
global using System.Security.Cryptography.X509Certificates;
global using Amazon.Keyspaces;
global using Amazon.Keyspaces.Model;
global using KeyspacesActions;
global using KeyspacesScenario;
global using Microsoft.Extensions.Configuration;
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;
global using Newtonsoft.Json;


namespace KeyspacesBasics;

/// <summary>
/// Amazon Keyspaces (for Apache Cassandra) scenario. Shows some of the basic
/// actions performed with Amazon Keyspaces.
/// </summary>
public class KeyspacesBasics
{
    private static ILogger logger = 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<IAmazonKeyspaces>()
            .AddTransient<KeyspacesWrapper>()
            .AddTransient<CassandraWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<KeyspacesBasics>();

        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 keyspacesWrapper = host.Services.GetRequiredService<KeyspacesWrapper>();
        var uiMethods = new UiMethods();

        var keyspaceName = configuration["KeyspaceName"];
        var tableName = configuration["TableName"];

        bool success; // Used to track the results of some operations.

        uiMethods.DisplayOverview();
        uiMethods.PressEnter();

        // Create the keyspace.
        var keyspaceArn = await keyspacesWrapper.CreateKeyspace(keyspaceName);

        // Wait for the keyspace to be available. GetKeyspace results in a
        // resource not found error until it is ready for use.
        try
        {
            var getKeyspaceArn = "";
            Console.Write($"Created {keyspaceName}. Waiting for it to become
available. ");
            do
            {
```

```
                getKeyspaceArn = await keyspacesWrapper.GetKeyspace(keyspaceName);
                Console.Write(". ");
            } while (getKeyspaceArn != keyspaceArn);
        }
        catch (ResourceNotFoundException)
        {
            Console.WriteLine("Waiting for keyspace to be created.");
        }

        Console.WriteLine($"\nThe keyspace {keyspaceName} is ready for use.");

        uiMethods.PressEnter();

        // Create the table.
        // First define the schema.
        var allColumns = new List<ColumnDefinition>
        {
            new ColumnDefinition { Name = "title", Type = "text" },
            new ColumnDefinition { Name = "year", Type = "int" },
            new ColumnDefinition { Name = "release_date", Type = "timestamp" },
            new ColumnDefinition { Name = "plot", Type = "text" },
        };

        var partitionKeys = new List<PartitionKey>
        {
            new PartitionKey { Name = "year", },
            new PartitionKey { Name = "title" },
        };

        var tableSchema = new SchemaDefinition
        {
            AllColumns = allColumns,
            PartitionKeys = partitionKeys,
        };

        var tableArn = await keyspacesWrapper.CreateTable(keyspaceName, tableSchema,
tableName);

        // Wait for the table to be active.
        try
        {
            var resp = new GetTableResponse();
            Console.Write("Waiting for the new table to be active. ");
            do
```

```
            {
                try
                {
                    resp = await keyspacesWrapper.GetTable(keyspaceName, tableName);
                    Console.Write(".");
                }
                catch (ResourceNotFoundException)
                {
                    Console.Write(".");
                }
            } while (resp.Status != TableStatus.ACTIVE);

            // Display the table's schema.
            Console.WriteLine($"\nTable {tableName} has been created in
{keyspaceName}");
            Console.WriteLine("Let's take a look at the schema.");
            uiMethods.DisplayTitle("All columns");
            resp.SchemaDefinition.AllColumns.ForEach(column =>
            {
                Console.WriteLine($"{column.Name,-40}\t{column.Type,-20}");
            });

            uiMethods.DisplayTitle("Cluster keys");
            resp.SchemaDefinition.ClusteringKeys.ForEach(clusterKey =>
            {

Console.WriteLine($"{clusterKey.Name,-40}\t{clusterKey.OrderBy,-20}");
            });

            uiMethods.DisplayTitle("Partition keys");
            resp.SchemaDefinition.PartitionKeys.ForEach(partitionKey =>
            {
                Console.WriteLine($"{partitionKey.Name}");
            });

            uiMethods.PressEnter();
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Error: {ex.Message}");
        }

        // Access Apache Cassandra using the Cassandra drive for C#.
        var cassandraWrapper = host.Services.GetRequiredService<CassandraWrapper>();
```

```
        var movieFilePath = configuration["MovieFile"];

        Console.WriteLine("Let's add some movies to the table we created.");
        var inserted = await cassandraWrapper.InsertIntoMovieTable(keyspaceName,
tableName, movieFilePath);

        uiMethods.PressEnter();

        Console.WriteLine("Added the following movies to the table:");
        var rows = await cassandraWrapper.GetMovies(keyspaceName, tableName);
        uiMethods.DisplayTitle("All Movies");

        foreach (var row in rows)
        {
            var title = row.GetValue<string>("title");
            var year = row.GetValue<int>("year");
            var plot = row.GetValue<string>("plot");
            var release_date = row.GetValue<DateTime>("release_date");
            Console.WriteLine($"{release_date}\t{title}\t{year}\n{plot}");
            Console.WriteLine(uiMethods.SepBar);
        }

        // Update the table schema
        uiMethods.DisplayTitle("Update table schema");
        Console.WriteLine("Now we will update the table to add a boolean field
called watched.");

        // First save the current time as a UTC Date so the original
        // table can be restored later.
        var timeChanged = DateTime.UtcNow;

        // Now update the schema.
        var resourceArn = await keyspacesWrapper.UpdateTable(keyspaceName,
tableName);
        uiMethods.PressEnter();

        Console.WriteLine("Now let's mark some of the movies as watched.");

        // Pick some files to mark as watched.
        var movieToWatch = rows[2].GetValue<string>("title");
        var watchedMovieYear = rows[2].GetValue<int>("year");
        var changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);
```

```
        movieToWatch = rows[6].GetValue<string>("title");
        watchedMovieYear = rows[6].GetValue<int>("year");
        changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);

        movieToWatch = rows[9].GetValue<string>("title");
        watchedMovieYear = rows[9].GetValue<int>("year");
        changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);

        movieToWatch = rows[10].GetValue<string>("title");
        watchedMovieYear = rows[10].GetValue<int>("year");
        changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);

        movieToWatch = rows[13].GetValue<string>("title");
        watchedMovieYear = rows[13].GetValue<int>("year");
        changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);

        uiMethods.DisplayTitle("Watched movies");
        Console.WriteLine("These movies have been marked as watched:");
        rows = await cassandraWrapper.GetWatchedMovies(keyspaceName, tableName);
        foreach (var row in rows)
        {
            var title = row.GetValue<string>("title");
            var year = row.GetValue<int>("year");
            Console.WriteLine($"{title,-40}\t{year,8}");
        }
        uiMethods.PressEnter();

        Console.WriteLine("We can restore the table to its previous state but that
can take up to 20 minutes to complete.");
        string answer;
        do
        {
            Console.WriteLine("Do you want to restore the table? (y/n)");
            answer = Console.ReadLine();
        } while (answer.ToLower() != "y" && answer.ToLower() != "n");

        if (answer == "y")
        {
            var restoredTableName = $"{tableName}_restored";
            var restoredTableArn = await keyspacesWrapper.RestoreTable(
```

```
                keyspaceName,
                tableName,
                restoredTableName,
                timeChanged);
            // Loop and call GetTable until the table is gone. Once it has been
            // deleted completely, GetTable will raise a ResourceNotFoundException.
            bool wasRestored = false;

            try
            {
                do
                {
                    var resp = await keyspacesWrapper.GetTable(keyspaceName,
restoredTableName);
                    wasRestored = (resp.Status == TableStatus.ACTIVE);
                } while (!wasRestored);
            }
            catch (ResourceNotFoundException)
            {
                // If the restored table raised an error, it isn't
                // ready yet.
                Console.Write(".");
            }
        }

        uiMethods.DisplayTitle("Clean up resources.");

        // Delete the table.
        success = await keyspacesWrapper.DeleteTable(keyspaceName, tableName);

        Console.WriteLine($"Table {tableName} successfully deleted from
{keyspaceName}.");
        Console.WriteLine("Waiting for the table to be removed completely. ");

        // Loop and call GetTable until the table is gone. Once it has been
        // deleted completely, GetTable will raise a ResourceNotFoundException.
        bool wasDeleted = false;

        try
        {
            do
            {
                var resp = await keyspacesWrapper.GetTable(keyspaceName, tableName);
            } while (!wasDeleted);
```

```
        }
        catch (ResourceNotFoundException ex)
        {
            wasDeleted = true;
            Console.WriteLine($"{ex.Message} indicates that the table has been
 deleted.");
        }

        // Delete the keyspace.
        success = await keyspacesWrapper.DeleteKeyspace(keyspaceName);
        Console.WriteLine("The keyspace has been deleted and the demo is now
 complete.");
    }
}
```

```
namespace KeyspacesActions;

/// <summary>
/// Performs Amazon Keyspaces (for Apache Cassandra) actions.
/// </summary>
public class KeyspacesWrapper
{
    private readonly IAmazonKeyspaces _amazonKeyspaces;

    /// <summary>
    /// Constructor for the KeyspaceWrapper.
    /// </summary>
    /// <param name="amazonKeyspaces">An Amazon Keyspaces client object.</param>
    public KeyspacesWrapper(IAmazonKeyspaces amazonKeyspaces)
    {
        _amazonKeyspaces = amazonKeyspaces;
    }

    /// <summary>
    /// Create a new keyspace.
    /// </summary>
    /// <param name="keyspaceName">The name for the new keyspace.</param>
    /// <returns>The Amazon Resource Name (ARN) of the new keyspace.</returns>
    public async Task<string> CreateKeyspace(string keyspaceName)
    {
        var response =
```

```
            await _amazonKeyspaces.CreateKeyspaceAsync(
                new CreateKeyspaceRequest { KeyspaceName = keyspaceName });
        return response.ResourceArn;
    }


    /// <summary>
    /// Create a new Amazon Keyspaces table.
    /// </summary>
    /// <param name="keyspaceName">The keyspace where the table will be created.</
param>
    /// <param name="schema">The schema for the new table.</param>
    /// <param name="tableName">The name of the new table.</param>
    /// <returns>The Amazon Resource Name (ARN) of the new table.</returns>
    public async Task<string> CreateTable(string keyspaceName, SchemaDefinition
 schema, string tableName)
    {
        var request = new CreateTableRequest
        {
            KeyspaceName = keyspaceName,
            SchemaDefinition = schema,
            TableName = tableName,
            PointInTimeRecovery = new PointInTimeRecovery { Status =
 PointInTimeRecoveryStatus.ENABLED }
        };

        var response = await _amazonKeyspaces.CreateTableAsync(request);
        return response.ResourceArn;
    }


    /// <summary>
    /// Delete an existing keyspace.
    /// </summary>
    /// <param name="keyspaceName"></param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteKeyspace(string keyspaceName)
    {
        var response = await _amazonKeyspaces.DeleteKeyspaceAsync(
            new DeleteKeyspaceRequest { KeyspaceName = keyspaceName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

```
    /// <summary>
    /// Delete an Amazon Keyspaces table.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteTable(string keyspaceName, string tableName)
    {
        var response = await _amazonKeyspaces.DeleteTableAsync(
            new DeleteTableRequest { KeyspaceName = keyspaceName, TableName =
tableName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Get data about a keyspace.
    /// </summary>
    /// <param name="keyspaceName">The name of the keyspace.</param>
    /// <returns>The Amazon Resource Name (ARN) of the keyspace.</returns>
    public async Task<string> GetKeyspace(string keyspaceName)
    {
        var response = await _amazonKeyspaces.GetKeyspaceAsync(
            new GetKeyspaceRequest { KeyspaceName = keyspaceName });
        return response.ResourceArn;
    }


    /// <summary>
    /// Get information about an Amazon Keyspaces table.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the Amazon Keyspaces table.</param>
    /// <returns>The response containing data about the table.</returns>
    public async Task<GetTableResponse> GetTable(string keyspaceName, string
tableName)
    {
        var response = await _amazonKeyspaces.GetTableAsync(
            new GetTableRequest { KeyspaceName = keyspaceName, TableName =
tableName });
        return response;
    }
```

```
    /// <summary>
    /// Lists all keyspaces for the account.
    /// </summary>
    /// <returns>Async task.</returns>
    public async Task ListKeyspaces()
    {
        var paginator = _amazonKeyspaces.Paginators.ListKeyspaces(new
ListKeyspacesRequest());

        Console.WriteLine("{0, -30}\t{1}", "Keyspace name", "Keyspace ARN");
        Console.WriteLine(new string('-', Console.WindowWidth));
        await foreach (var keyspace in paginator.Keyspaces)
        {

Console.WriteLine($"{keyspace.KeyspaceName,-30}\t{keyspace.ResourceArn}");
        }
    }


    /// <summary>
    /// Lists the Amazon Keyspaces tables in a keyspace.
    /// </summary>
    /// <param name="keyspaceName">The name of the keyspace.</param>
    /// <returns>A list of TableSummary objects.</returns>
    public async Task<List<TableSummary>> ListTables(string keyspaceName)
    {
        var response = await _amazonKeyspaces.ListTablesAsync(new ListTablesRequest
{ KeyspaceName = keyspaceName });
        response.Tables.ForEach(table =>
        {

Console.WriteLine($"{table.KeyspaceName}\t{table.TableName}\t{table.ResourceArn}");
        });

        return response.Tables;
    }


    /// <summary>
    /// Restores the specified table to the specified point in time.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table to restore.</param>
```

```
    /// <param name="timestamp">The time to which the table will be restored.</
param>
    /// <returns>The Amazon Resource Name (ARN) of the restored table.</returns>
    public async Task<string> RestoreTable(string keyspaceName, string tableName,
 string restoredTableName, DateTime timestamp)
    {
        var request = new RestoreTableRequest
        {
            RestoreTimestamp = timestamp,
            SourceKeyspaceName = keyspaceName,
            SourceTableName = tableName,
            TargetKeyspaceName = keyspaceName,
            TargetTableName = restoredTableName
        };

        var response = await _amazonKeyspaces.RestoreTableAsync(request);
        return response.RestoredTableARN;
    }


    /// <summary>
    /// Updates the movie table to add a boolean column named watched.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table to change.</param>
    /// <returns>The Amazon Resource Name (ARN) of the updated table.</returns>
    public async Task<string> UpdateTable(string keyspaceName, string tableName)
    {
        var newColumn = new ColumnDefinition { Name = "watched", Type = "boolean" };
        var request = new UpdateTableRequest
        {
            KeyspaceName = keyspaceName,
            TableName = tableName,
            AddColumns = new List<ColumnDefinition> { newColumn }
        };
        var response = await _amazonKeyspaces.UpdateTableAsync(request);
        return response.ResourceArn;
    }

}
```

```
using System.Net;
using Cassandra;

namespace KeyspacesScenario;

/// <summary>
/// Class to perform CRUD methods on an Amazon Keyspaces (for Apache Cassandra)
 database.
///
/// NOTE: This sample uses a plain text authenticator for example purposes only.
/// Recommended best practice is to use a SigV4 authentication plugin, if available.
/// </summary>
public class CassandraWrapper
{
    private readonly IConfiguration _configuration;
    private readonly string _localPathToFile;
    private const string _certLocation = "https://certs.secureserver.net/repository/
sf-class2-root.crt";
    private const string _certFileName = "sf-class2-root.crt";
    private readonly X509Certificate2Collection _certCollection;
    private X509Certificate2 _amazoncert;
    private Cluster _cluster;

    // User name and password for the service.
    private string _userName = null!;
    private string _pwd = null!;

    public CassandraWrapper()
    {
        _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();

        _localPathToFile = Path.GetTempPath();

        // Get the Starfield digital certificate and save it locally.
        var client = new WebClient();
        client.DownloadFile(_certLocation, $"{_localPathToFile}/{_certFileName}");

        //var httpClient = new HttpClient();
```

```
        //var httpResult = httpClient.Get(fileUrl);
        //using var resultStream = await httpResult.Content.ReadAsStreamAsync();
        //using var fileStream = File.Create(pathToSave);
        //resultStream.CopyTo(fileStream);

        _certCollection = new X509Certificate2Collection();
        _amazoncert = new X509Certificate2($"{_localPathToFile}/{_certFileName}");

        // Get the user name and password stored in the configuration file.
        _userName = _configuration["UserName"]!;
        _pwd = _configuration["Password"]!;

        // For a list of Service Endpoints for Amazon Keyspaces, see:
        // https://docs.aws.amazon.com/keyspaces/latest/devguide/
programmatic.endpoints.html
        var awsEndpoint = _configuration["ServiceEndpoint"];

        _cluster = Cluster.Builder()
            .AddContactPoints(awsEndpoint)
            .WithPort(9142)
            .WithAuthProvider(new PlainTextAuthProvider(_userName, _pwd))
            .WithSSL(new SSLOptions().SetCertificateCollection(_certCollection))
            .WithQueryOptions(
                new QueryOptions()
                    .SetConsistencyLevel(ConsistencyLevel.LocalQuorum)
                    .SetSerialConsistencyLevel(ConsistencyLevel.LocalSerial))
            .Build();
    }

    /// <summary>
    /// Loads the contents of a JSON file into a list of movies to be
    /// added to the Apache Cassandra table.
    /// </summary>
    /// <param name="movieFileName">The full path to the JSON file.</param>
    /// <returns>A list of movie objects.</returns>
    public List<Movie> ImportMoviesFromJson(string movieFileName, int numToImport =
 0)
    {
        if (!File.Exists(movieFileName))
        {
            return null!;
        }

        using var sr = new StreamReader(movieFileName);
```

```
        string json = sr.ReadToEnd();

        var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

        // If numToImport = 0, return all movies in the collection.
        if (numToImport == 0)
        {
            // Now return the entire list of movies.
            return allMovies;
        }
        else
        {
            // Now return the first numToImport entries.
            return allMovies.GetRange(0, numToImport);
        }
    }

    /// <summary>
    /// Insert movies into the movie table.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="movieTableName">The Amazon Keyspaces table.</param>
    /// <param name="movieFilePath">The path to the resource file containing
    /// movie data to insert into the table.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> InsertIntoMovieTable(string keyspaceName, string
movieTableName, string movieFilePath, int numToImport = 20)
    {
        // Get some movie data from the movies.json file
        var movies = ImportMoviesFromJson(movieFilePath, numToImport);

        var session = _cluster.Connect(keyspaceName);

        string insertCql;

        RowSet rs;

        // Now we insert the numToImport movies into the table.
        foreach (var movie in movies)
        {
            // Escape single quote characters in the plot.
            insertCql = $"INSERT INTO {keyspaceName}.{movieTableName}
(title, year, release_date, plot) values($${movie.Title}$$, {movie.Year},
'{movie.Info.Release_Date.ToString("yyyy-MM-dd")}', $${movie.Info.Plot}$$)";
```

```
            rs = await session.ExecuteAsync(new SimpleStatement(insertCql));
        }

        return true;
    }

    /// <summary>
    /// Gets all of the movies in the movies table.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table.</param>
    /// <returns>A list of row objects containing movie data.</returns>
    public async Task<List<Row>> GetMovies(string keyspaceName, string tableName)
    {
        var session = _cluster.Connect();
        RowSet rs;
        try
        {
            rs = await session.ExecuteAsync(new SimpleStatement($"SELECT * FROM
{keyspaceName}.{tableName}"));

            // Extract the row data from the returned RowSet.
            var rows = rs.GetRows().ToList();
            return rows;
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
            return null!;
        }
    }

    /// <summary>
    /// Mark a movie in the movie table as watched.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table.</param>
    /// <param name="title">The title of the movie to mark as watched.</param>
    /// <param name="year">The year the movie was released.</param>
    /// <returns>A set of rows containing the changed data.</returns>
    public async Task<List<Row>> MarkMovieAsWatched(string keyspaceName, string
tableName, string title, int year)
    {
        var session = _cluster.Connect();
```

```
        string updateCql = $"UPDATE {keyspaceName}.{tableName} SET watched=true
 WHERE title = $${title}$$ AND year = {year};";
        var rs = await session.ExecuteAsync(new SimpleStatement(updateCql));
        var rows = rs.GetRows().ToList();
        return rows;
    }

    /// <summary>
    /// Retrieve the movies in the movies table where watched is true.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table.</param>
    /// <returns>A list of row objects containing information about movies
    /// where watched is true.</returns>
    public async Task<List<Row>> GetWatchedMovies(string keyspaceName, string
 tableName)
    {
        var session = _cluster.Connect();
        RowSet rs;
        try
        {
            rs = await session.ExecuteAsync(new SimpleStatement($"SELECT title,
 year, plot FROM {keyspaceName}.{tableName} WHERE watched = true ALLOW FILTERING"));

            // Extract the row data from the returned RowSet.
            var rows = rs.GetRows().ToList();
            return rows;
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
            return null!;
        }
    }
}
```

- API の詳細については、『AWS SDK for .NET API リファレンス』の以下のトピックを参照してください。
  - CreateKeyspace
  - CreateTable

- DeleteKeyspace
- DeleteTable
- GetKeyspace
- GetTable
- ListKeyspaces
- ListTables
- RestoreTable
- UpdateTable

を使用した Kinesis の例 AWS SDK for .NET

次のコード例は、Kinesis AWS SDK for .NET で 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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。このリンクには GitHub、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が記載されています。

トピック

- アクション
- サーバーレスサンプル

アクション

`AddTagsToStream`

次の例は、`AddTagsToStream` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// This example shows how to apply key/value pairs to an Amazon Kinesis
/// stream.
/// </summary>
public class TagStream
{
    public static async Task Main()
    {
        IAmazonKinesis client = new AmazonKinesisClient();

        string streamName = "AmazonKinesisStream";
        var tags = new Dictionary<string, string>
        {
            { "Project", "Sample Kinesis Project" },
            { "Application", "Sample Kinesis App" },
        };

        var success = await ApplyTagsToStreamAsync(client, streamName, tags);

        if (success)
        {
            Console.WriteLine($"Taggs successfully added to {streamName}.");
        }
        else
        {
            Console.WriteLine("Tags were not added to the stream.");
        }
    }
```

```
        /// <summary>
        /// Applies the set of tags to the named Kinesis stream.
        /// </summary>
        /// <param name="client">The initialized Kinesis client.</param>
        /// <param name="streamName">The name of the Kinesis stream to which
        /// the tags will be attached.</param>
        /// <param name="tags">A sictionary containing key/value pairs which
        /// will be used to create the Kinesis tags.</param>
        /// <returns>A Boolean value which represents the success or failure
        /// of AddTagsToStreamAsync.</returns>
        public static async Task<bool> ApplyTagsToStreamAsync(
            IAmazonKinesis client,
            string streamName,
            Dictionary<string, string> tags)
        {
            var request = new AddTagsToStreamRequest
            {
                StreamName = streamName,
                Tags = tags,
            };

            var response = await client.AddTagsToStreamAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
    }
```

- API の詳細については、「 API リファレンスAddTagsToStream」の「」を参照してください。AWS SDK for .NET

CreateStream

次の例は、CreateStream を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

    /// <summary>
    /// This example shows how to create a new Amazon Kinesis stream.
    /// </summary>
    public class CreateStream
    {
        public static async Task Main()
        {
            IAmazonKinesis client = new AmazonKinesisClient();

            string streamName = "AmazonKinesisStream";
            int shardCount = 1;

            var success = await CreateNewStreamAsync(client, streamName,
shardCount);
            if (success)
            {
                Console.WriteLine($"The stream, {streamName} successfully
created.");
            }
        }

        /// <summary>
        /// Creates a new Kinesis stream.
        /// </summary>
        /// <param name="client">An initialized Kinesis client.</param>
        /// <param name="streamName">The name for the new stream.</param>
        /// <param name="shardCount">The number of shards the new stream will
        /// use. The throughput of the stream is a function of the number of
        /// shards; more shards are required for greater provisioned
```

```
        /// throughput.</param>
        /// <returns>A Boolean value indicating whether the stream was created.</
returns>
        public static async Task<bool> CreateNewStreamAsync(IAmazonKinesis client,
 string streamName, int shardCount)
        {
            var request = new CreateStreamRequest
            {
                StreamName = streamName,
                ShardCount = shardCount,
            };

            var response = await client.CreateStreamAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
    }
```

- API の詳細については、「 API リファレンスCreateStream」の「」を参照してください。 AWS SDK for .NET

`DeleteStream`

次の例は、`DeleteStream` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

    /// <summary>
```

```
    /// Shows how to delete an Amazon Kinesis stream.
    /// </summary>
    public class DeleteStream
    {
        public static async Task Main()
        {
            IAmazonKinesis client = new AmazonKinesisClient();
            string streamName = "AmazonKinesisStream";

            var success = await DeleteStreamAsync(client, streamName);

            if (success)
            {
                Console.WriteLine($"Stream, {streamName} successfully deleted.");
            }
            else
            {
                Console.WriteLine("Stream not deleted.");
            }
        }

        /// <summary>
        /// Deletes a Kinesis stream.
        /// </summary>
        /// <param name="client">An initialized Kinesis client object.</param>
        /// <param name="streamName">The name of the string to delete.</param>
        /// <returns>A Boolean value representing the success of the operation.</
returns>
        public static async Task<bool> DeleteStreamAsync(IAmazonKinesis client,
 string streamName)
        {
            // If EnforceConsumerDeletion is true, any consumers
            // of this stream will also be deleted. If it is set
            // to false and this stream has any consumers, the
            // call will fail with a ResourceInUseException.
            var request = new DeleteStreamRequest
            {
                StreamName = streamName,
                EnforceConsumerDeletion = true,
            };

            var response = await client.DeleteStreamAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
```

```
        }
    }
```

- API の詳細については、「 API リファレンスDeleteStream」の「」を参照してください。 AWS SDK for .NET

`DeregisterStreamConsumer`

次の例は、DeregisterStreamConsumer を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

    /// <summary>
    /// Shows how to deregister a consumer from an Amazon Kinesis stream.
    /// </summary>
    public class DeregisterConsumer
    {
        public static async Task Main(string[] args)
        {
            IAmazonKinesis client = new AmazonKinesisClient();

            string streamARN = "arn:aws:kinesis:us-west-2:000000000000:stream/
AmazonKinesisStream";
            string consumerName = "CONSUMER_NAME";
            string consumerARN = "arn:aws:kinesis:us-west-2:000000000000:stream/
AmazonKinesisStream/consumer/CONSUMER_NAME:000000000000";
```

```
            var success = await DeregisterConsumerAsync(client, streamARN,
 consumerARN, consumerName);

            if (success)
            {
                Console.WriteLine($"{consumerName} successfully deregistered.");
            }
            else
            {
                Console.WriteLine($"{consumerName} was not successfully
 deregistered.");
            }
        }

        /// <summary>
        /// Deregisters a consumer from a Kinesis stream.
        /// </summary>
        /// <param name="client">An initialized Kinesis client object.</param>
        /// <param name="streamARN">The ARN of a Kinesis stream.</param>
        /// <param name="consumerARN">The ARN of the consumer.</param>
        /// <param name="consumerName">The name of the consumer.</param>
        /// <returns>A Boolean value representing the success of the operation.</
returns>
        public static async Task<bool> DeregisterConsumerAsync(
            IAmazonKinesis client,
            string streamARN,
            string consumerARN,
            string consumerName)
        {
            var request = new DeregisterStreamConsumerRequest
            {
                StreamARN = streamARN,
                ConsumerARN = consumerARN,
                ConsumerName = consumerName,
            };

            var response = await client.DeregisterStreamConsumerAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
    }
```

- API の詳細については、「 API リファレンスDeregisterStreamConsumer」の「」を参照してください。 AWS SDK for .NET

`ListStreamConsumers`

次の例は、`ListStreamConsumers` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon.Kinesis;
    using Amazon.Kinesis.Model;

    /// <summary>
    /// List the consumers of an Amazon Kinesis stream.
    /// </summary>
    public class ListConsumers
    {
        public static async Task Main()
        {
            IAmazonKinesis client = new AmazonKinesisClient();

            string streamARN = "arn:aws:kinesis:us-east-2:000000000000:stream/
AmazonKinesisStream";
            int maxResults = 10;

            var consumers = await ListConsumersAsync(client, streamARN, maxResults);

            if (consumers.Count > 0)
            {
                consumers
                    .ForEach(c => Console.WriteLine($"Name: {c.ConsumerName} ARN:
 {c.ConsumerARN}"));
            }
```

```
            else
            {
                Console.WriteLine("No consumers found.");
            }
        }

        /// <summary>
        /// Retrieve a list of the consumers for a Kinesis stream.
        /// </summary>
        /// <param name="client">An initialized Kinesis client object.</param>
        /// <param name="streamARN">The ARN of the stream for which we want to
        /// retrieve a list of clients.</param>
        /// <param name="maxResults">The maximum number of results to return.</
param>
        /// <returns>A list of Consumer objects.</returns>
        public static async Task<List<Consumer>> ListConsumersAsync(IAmazonKinesis
 client, string streamARN, int maxResults)
        {
            var request = new ListStreamConsumersRequest
            {
                StreamARN = streamARN,
                MaxResults = maxResults,
            };

            var response = await client.ListStreamConsumersAsync(request);

            return response.Consumers;
        }
    }
```

- API の詳細については、「 API リファレンスListStreamConsumers」の「」を参照してください。 AWS SDK for .NET

ListStreams

次の例は、ListStreams を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// Retrieves and displays a list of existing Amazon Kinesis streams.
/// </summary>
public class ListStreams
{
    public static async Task Main(string[] args)
    {
        IAmazonKinesis client = new AmazonKinesisClient();
        var response = await client.ListStreamsAsync(new ListStreamsRequest());

        List<string> streamNames = response.StreamNames;

        if (streamNames.Count > 0)
        {
            streamNames
                .ForEach(s => Console.WriteLine($"Stream name: {s}"));
        }
        else
        {
            Console.WriteLine("No streams were found.");
        }
    }
}
```

- API の詳細については、「 API リファレンスListStreams」の「」を参照してください。 AWS SDK for .NET

ListTagsForStream

次の例は、ListTagsForStream を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon.Kinesis;
    using Amazon.Kinesis.Model;

    /// <summary>
    /// Shows how to list the tags that have been attached to an Amazon Kinesis
    /// stream.
    /// </summary>
    public class ListTags
    {
        public static async Task Main()
        {
            IAmazonKinesis client = new AmazonKinesisClient();
            string streamName = "AmazonKinesisStream";

            await ListTagsAsync(client, streamName);
        }

        /// <summary>
        /// List the tags attached to a Kinesis stream.
        /// </summary>
        /// <param name="client">An initialized Kinesis client object.</param>
        /// <param name="streamName">The name of the Kinesis stream for which you
        /// wish to display tags.</param>
        public static async Task ListTagsAsync(IAmazonKinesis client, string
streamName)
        {
            var request = new ListTagsForStreamRequest
            {
```

```
                StreamName = streamName,
                Limit = 10,
            };

            var response = await client.ListTagsForStreamAsync(request);
            DisplayTags(response.Tags);

            while (response.HasMoreTags)
            {
                request.ExclusiveStartTagKey = response.Tags[response.Tags.Count -
1].Key;
                response = await client.ListTagsForStreamAsync(request);
            }
        }

        /// <summary>
        /// Displays the items in a list of Kinesis tags.
        /// </summary>
        /// <param name="tags">A list of the Tag objects to be displayed.</param>
        public static void DisplayTags(List<Tag> tags)
        {
            tags
                .ForEach(t => Console.WriteLine($"Key: {t.Key} Value: {t.Value}"));
        }
    }
```

- API の詳細については、「 API リファレンスListTagsForStream」の「」を参照してください。 AWS SDK for .NET

`RegisterStreamConsumer`

次の例は、`RegisterStreamConsumer` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

    /// <summary>
    /// This example shows how to register a consumer to an Amazon Kinesis
    /// stream.
    /// </summary>
    public class RegisterConsumer
    {
        public static async Task Main()
        {
            IAmazonKinesis client = new AmazonKinesisClient();
            string consumerName = "NEW_CONSUMER_NAME";
            string streamARN = "arn:aws:kinesis:us-east-2:000000000000:stream/
AmazonKinesisStream";

            var consumer = await RegisterConsumerAsync(client, consumerName,
 streamARN);

            if (consumer is not null)
            {
                Console.WriteLine($"{consumer.ConsumerName}");
            }
        }

        /// <summary>
        /// Registers the consumer to a Kinesis stream.
        /// </summary>
        /// <param name="client">The initialized Kinesis client object.</param>
        /// <param name="consumerName">A string representing the consumer.</param>
        /// <param name="streamARN">The ARN of the stream.</param>
        /// <returns>A Consumer object that contains information about the
 consumer.</returns>
        public static async Task<Consumer> RegisterConsumerAsync(IAmazonKinesis
 client, string consumerName, string streamARN)
        {
            var request = new RegisterStreamConsumerRequest
            {
                ConsumerName = consumerName,
                StreamARN = streamARN,
            };
```

```
            var response = await client.RegisterStreamConsumerAsync(request);
            return response.Consumer;
        }
    }
```

- API の詳細については、「 API リファレンスRegisterStreamConsumer」の「」を参照してください。 AWS SDK for .NET

サーバーレスサンプル

Kinesis トリガーから Lambda 関数を呼び出す

次のコード例では、Kinesis ストリームからレコードを受信することによってトリガーされるイベントを受け取る、Lambda 関数の実装方法を示しています。この関数は Kinesis ペイロードを取得し、それを Base64 からデコードして、そのレコードの内容を口グ記録します。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

.NET を使用した Lambda での Kinesis イベントの消費。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali
```

```
namespace KinesisIntegrationSampleCode;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task FunctionHandler(KinesisEvent evnt, ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return;
        }

        foreach (var record in evnt.Records)
        {
            try
            {
                Logger.LogInformation($"Processed Event with EventId:
 {record.EventId}");
                string data = await GetRecordDataAsync(record.Kinesis, context);
                Logger.LogInformation($"Data: {data}");
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex)
            {
                Logger.LogError($"An error occurred {ex.Message}");
                throw;
            }
        }
        Logger.LogInformation($"Successfully processed {evnt.Records.Count}
 records.");
    }

    private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
 ILambdaContext context)
    {
        byte[] bytes = record.Data.ToArray();
        string data = Encoding.UTF8.GetString(bytes);
        await Task.CompletedTask; //Placeholder for actual async work
        return data;
    }
```

```
}
```

Kinesis トリガーを使用した Lambda 関数でのバッチアイテムの失敗のレポート

以下のコード例では、Kinesis ストリームからイベントを受け取る Lambda 関数のための、部分的なバッチレスポンスの実装方法を示しています。この関数は、レスポンスとしてバッチアイテムの失敗を報告し、対象のメッセージを後で再試行するよう Lambda に伝えます。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

.NET を使用した Lambda での Kinesis バッチアイテム失敗のレポート。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using System.Text.Json.Serialization;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace KinesisIntegration;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task<StreamsEventResponse> FunctionHandler(KinesisEvent evnt,
 ILambdaContext context)
    {
```

```
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return new StreamsEventResponse();
        }

        foreach (var record in evnt.Records)
        {
            try
            {
                Logger.LogInformation($"Processed Event with EventId:
{record.EventId}");
                string data = await GetRecordDataAsync(record.Kinesis, context);
                Logger.LogInformation($"Data: {data}");
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex)
            {
                Logger.LogError($"An error occurred {ex.Message}");
                /* Since we are working with streams, we can return the failed item
immediately.
                   Lambda will immediately begin to retry processing from this
failed item onwards. */
                return new StreamsEventResponse
                {
                    BatchItemFailures = new
List<StreamsEventResponse.BatchItemFailure>
                    {
                        new StreamsEventResponse.BatchItemFailure { ItemIdentifier =
record.Kinesis.SequenceNumber }
                    }
                };
            }
        }
        Logger.LogInformation($"Successfully processed {evnt.Records.Count}
records.");
        return new StreamsEventResponse();
    }

    private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
ILambdaContext context)
    {
        byte[] bytes = record.Data.ToArray();
        string data = Encoding.UTF8.GetString(bytes);
```

```
        await Task.CompletedTask; //Placeholder for actual async work
        return data;
    }
}

public class StreamsEventResponse
{
    [JsonPropertyName("batchItemFailures")]
    public IList<BatchItemFailure> BatchItemFailures { get; set; }
    public class BatchItemFailure
    {
        [JsonPropertyName("itemIdentifier")]
        public string ItemIdentifier { get; set; }
    }
}
```

AWS KMS を使用した の例 AWS SDK for .NET

次のコード例は、 AWS SDK for .NET で 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 KMS。

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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。このリンクには GitHub、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が記載されています。

トピック

- アクション

アクション

`CreateAlias`

次の例は、`CreateAlias` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

/// <summary>
/// Creates an alias for an AWS Key Management Service (AWS KMS) key.
/// </summary>
public class CreateAlias
{
    public static async Task Main()
    {
        var client = new AmazonKeyManagementServiceClient();

        // The alias name must start with alias/ and can be
        // up to 256 alphanumeric characters long.
        var aliasName = "alias/ExampleAlias";

        // The value supplied as the TargetKeyId can be either
        // the key ID or key Amazon Resource Name (ARN) of the
        // AWS KMS key.
        var keyId = "1234abcd-12ab-34cd-56ef-1234567890ab";

        var request = new CreateAliasRequest
        {
            AliasName = aliasName,
            TargetKeyId = keyId,
        };

        var response = await client.CreateAliasAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"Alias, {aliasName}, successfully created.");
```

```
            }
            else
            {
                Console.WriteLine($"Could not create alias.");
            }
        }
    }
```

- API の詳細については、「 API リファレンスCreateAlias」の「」を参照してください。 AWS SDK for .NET

`CreateGrant`

次の例は、CreateGrant を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        public static async Task Main()
        {
            var client = new AmazonKeyManagementServiceClient();

            // The identity that is given permission to perform the operations
            // specified in the grant.
            var grantee = "arn:aws:iam::111122223333:role/ExampleRole";

            // The identifier of the AWS KMS key to which the grant applies. You
            // can use the key ID or the Amazon Resource Name (ARN) of the KMS key.
            var keyId = "7c9eccc2-38cb-4c4f-9db3-766ee8dd3ad4";

            var request = new CreateGrantRequest
            {
                GranteePrincipal = grantee,
                KeyId = keyId,
```

```
                // A list of operations that the grant allows.
                Operations = new List<string>
                {
                    "Encrypt",
                    "Decrypt",
                },
            };

            var response = await client.CreateGrantAsync(request);

            string grantId = response.GrantId; // The unique identifier of the
grant.
            string grantToken = response.GrantToken; // The grant token.

            Console.WriteLine($"Id: {grantId}, Token: {grantToken}");
        }
    }
```

- API の詳細については、「 API リファレンスCreateGrant」の「」を参照してください。 AWS SDK for .NET

`CreateKey`

次の例は、CreateKey を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

```
    /// <summary>
    /// Shows how to create a new AWS Key Management Service (AWS KMS)
    /// key.
    /// </summary>
    public class CreateKey
    {
        public static async Task Main()
        {
            // Note that if you need to create a Key in an AWS Region
            // other than the Region defined for the default user, you need to
            // pass the Region to the client constructor.
            var client = new AmazonKeyManagementServiceClient();

            // The call to CreateKeyAsync will create a symmetrical AWS KMS
            // key. For more information about symmetrical and asymmetrical
            // keys, see:
            //
            // https://docs.aws.amazon.com/kms/latest/developerguide/symm-asymm-
choose.html
            var response = await client.CreateKeyAsync(new CreateKeyRequest());

            // The KeyMetadata object contains information about the new AWS KMS
 key.
            KeyMetadata keyMetadata = response.KeyMetadata;

            if (keyMetadata is not null)
            {
                Console.WriteLine($"KMS Key: {keyMetadata.KeyId} was successfully
 created.");
            }
            else
            {
                Console.WriteLine("Could not create KMS Key.");
            }
        }
    }
```

- API の詳細については、「 API リファレンスCreateKey」の「」を参照してください。 AWS SDK for .NET

DescribeKey

次の例は、DescribeKey を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

    /// <summary>
    /// Retrieve information about an AWS Key Management Service (AWS KMS) key.
    /// You can supply either the key Id or the key Amazon Resource Name (ARN)
    /// to the DescribeKeyRequest KeyId property.
    /// </summary>
    public class DescribeKey
    {
        public static async Task Main()
        {
            var keyId = "7c9eccc2-38cb-4c4f-9db3-766ee8dd3ad4";
            var request = new DescribeKeyRequest
            {
                KeyId = keyId,
            };

            var client = new AmazonKeyManagementServiceClient();

            var response = await client.DescribeKeyAsync(request);
            var metadata = response.KeyMetadata;

            Console.WriteLine($"{metadata.KeyId} created on:
{metadata.CreationDate}");
            Console.WriteLine($"State: {metadata.KeyState}");
            Console.WriteLine($"{metadata.Description}");
        }
    }
```

- API の詳細については、「 API リファレンスDescribeKey」の「」を参照してください。 AWS SDK for .NET

`DisableKey`

次の例は、DisableKey を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

/// <summary>
/// Disable an AWS Key Management Service (AWS KMS) key and then retrieve
/// the key's status to show that it has been disabled.
/// </summary>
public class DisableKey
{
    public static async Task Main()
    {
        var client = new AmazonKeyManagementServiceClient();

        // The identifier of the AWS KMS key to disable. You can use the
        // key Id or the Amazon Resource Name (ARN) of the AWS KMS key.
        var keyId = "1234abcd-12ab-34cd-56ef-1234567890ab";

        var request = new DisableKeyRequest
        {
            KeyId = keyId,
        };
```

```
            var response = await client.DisableKeyAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                // Retrieve information about the key to show that it has now
                // been disabled.
                var describeResponse = await client.DescribeKeyAsync(new
DescribeKeyRequest
                {
                    KeyId = keyId,
                });
                Console.WriteLine($"{describeResponse.KeyMetadata.KeyId} - state:
{describeResponse.KeyMetadata.KeyState}");
            }
        }
    }
```

- API の詳細については、「 API リファレンスDisableKey」の「」を参照してください。 AWS SDK for .NET

`EnableKey`

次の例は、EnableKey を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

    /// <summary>
```

```
    /// Enable an AWS Key Management Service (AWS KMS) key.
    /// </summary>
    public class EnableKey
    {
        public static async Task Main()
        {
            var client = new AmazonKeyManagementServiceClient();

            // The identifier of the AWS KMS key to enable. You can use the
            // key Id or the Amazon Resource Name (ARN) of the AWS KMS key.
            var keyId = "1234abcd-12ab-34cd-56ef-1234567890ab";

            var request = new EnableKeyRequest
            {
                KeyId = keyId,
            };

            var response = await client.EnableKeyAsync(request);
            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                // Retrieve information about the key to show that it has now
                // been enabled.
                var describeResponse = await client.DescribeKeyAsync(new
DescribeKeyRequest
                {
                    KeyId = keyId,
                });
                Console.WriteLine($"{describeResponse.KeyMetadata.KeyId} - state:
{describeResponse.KeyMetadata.KeyState}");
            }
        }
    }
```

- API の詳細については、「 API リファレンスEnableKey」の「」を参照してください。 AWS SDK for .NET

ListAliases

次の例は、ListAliases を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

   /// <summary>
   /// List the AWS Key Management Service (AWS KMS) aliases that have been defined
for
   /// the keys in the same AWS Region as the default user. If you want to list
   /// the aliases in a different Region, pass the Region to the client
   /// constructor.
   /// </summary>
   public class ListAliases
   {
       public static async Task Main()
       {
           var client = new AmazonKeyManagementServiceClient();
           var request = new ListAliasesRequest();
           var response = new ListAliasesResponse();

           do
           {
               response = await client.ListAliasesAsync(request);

               response.Aliases.ForEach(alias =>
               {
                   Console.WriteLine($"Created: {alias.CreationDate} Last Update:
{alias.LastUpdatedDate} Name: {alias.AliasName}");
               });

               request.Marker = response.NextMarker;
           }
           while (response.Truncated);
       }
```

```
    }
```

- API の詳細については、「 API リファレンスListAliases」の「」を参照してください。 AWS SDK for .NET

`ListGrants`

次の例は、`ListGrants` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

    /// <summary>
    /// List the AWS Key Management Service (AWS KMS) grants that are associated
with
    /// a specific key.
    /// </summary>
    public class ListGrants
    {
        public static async Task Main()
        {
            // The identifier of the AWS KMS key to disable. You can use the
            // key Id or the Amazon Resource Name (ARN) of the AWS KMS key.
            var keyId = "1234abcd-12ab-34cd-56ef-1234567890ab";
            var client = new AmazonKeyManagementServiceClient();
            var request = new ListGrantsRequest
            {
                KeyId = keyId,
            };
```

```
            var response = new ListGrantsResponse();

            do
            {
                response = await client.ListGrantsAsync(request);

                response.Grants.ForEach(grant =>
                {
                    Console.WriteLine($"{grant.GrantId}");
                });

                request.Marker = response.NextMarker;
            }
            while (response.Truncated);
        }
    }
```

- API の詳細については、「 API リファレンスListGrants」の「」を参照してください。 AWS SDK for .NET

`ListKeys`

次の例は、ListKeys を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

    /// <summary>
```

```
    /// List the AWS Key Managements Service (AWS KMS) keys for the AWS Region
    /// of the default user. To list keys in another AWS Region, supply the Region
    /// as a parameter to the client constructor.
    /// </summary>
    public class ListKeys
    {
        public static async Task Main()
        {
            var client = new AmazonKeyManagementServiceClient();
            var request = new ListKeysRequest();
            var response = new ListKeysResponse();

            do
            {
                response = await client.ListKeysAsync(request);

                response.Keys.ForEach(key =>
                {
                    Console.WriteLine($"ID: {key.KeyId}, {key.KeyArn}");
                });

                // Set the Marker property when response.Truncated is true
                // in order to get the next keys.
                request.Marker = response.NextMarker;
            }
            while (response.Truncated);
        }
    }
```

- API の詳細については、「 API リファレンスListKeys」の「」を参照してください。 AWS SDK for .NET

を使用した Lambda の例 AWS SDK for .NET

次のコード例は、Lambda AWS SDK for .NET で 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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。このリンクには GitHub、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が記載されています。

開始方法

Hello Lambda

次のコード例では、Lambda の使用を開始する方法について示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
namespace LambdaActions;

using Amazon.Lambda;

public class HelloLambda
{
    static async Task Main(string[] args)
    {
        var lambdaClient = new AmazonLambdaClient();

        Console.WriteLine("Hello AWS Lambda");
        Console.WriteLine("Let's get started with AWS Lambda by listing your
 existing Lambda functions:");

        var response = await lambdaClient.ListFunctionsAsync();
        response.Functions.ForEach(function =>
        {
            Console.WriteLine($"{function.FunctionName}\t{function.Description}");
        });
    }
}
```

- API の詳細については、「 API リファレンスListFunctions」の「」を参照してください。 AWS SDK for .NET

トピック

- アクション
- シナリオ
- サーバーレスサンプル

アクション

`CreateFunction`

次の例は、CreateFunction を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
/// Creates a new Lambda function.
/// </summary>
/// <param name="functionName">The name of the function.</param>
/// <param name="s3Bucket">The Amazon Simple Storage Service (Amazon S3)
/// bucket where the zip file containing the code is located.</param>
/// <param name="s3Key">The Amazon S3 key of the zip file.</param>
/// <param name="role">The Amazon Resource Name (ARN) of a role with the
/// appropriate Lambda permissions.</param>
/// <param name="handler">The name of the handler function.</param>
/// <returns>The Amazon Resource Name (ARN) of the newly created
/// Lambda function.</returns>
public async Task<string> CreateLambdaFunctionAsync(
    string functionName,
    string s3Bucket,
```

```
        string s3Key,
        string role,
        string handler)
    {
        // Defines the location for the function code.
        // S3Bucket - The S3 bucket where the file containing
        //            the source code is stored.
        // S3Key    - The name of the file containing the code.
        var functionCode = new FunctionCode
        {
            S3Bucket = s3Bucket,
            S3Key = s3Key,
        };

        var createFunctionRequest = new CreateFunctionRequest
        {
            FunctionName = functionName,
            Description = "Created by the Lambda .NET API",
            Code = functionCode,
            Handler = handler,
            Runtime = Runtime.Dotnet6,
            Role = role,
        };

        var reponse = await
_lambdaService.CreateFunctionAsync(createFunctionRequest);
        return reponse.FunctionArn;
    }
```

- API の詳細については、「 API リファレンスCreateFunction」の「」を参照してください。AWS SDK for .NET

DeleteFunction

次の例は、DeleteFunction を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete an AWS Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the Lambda function to
    /// delete.</param>
    /// <returns>A Boolean value that indicates the success of the action.</returns>
    public async Task<bool> DeleteFunctionAsync(string functionName)
    {
        var request = new DeleteFunctionRequest
        {
            FunctionName = functionName,
        };

        var response = await _lambdaService.DeleteFunctionAsync(request);

        // A return value of NoContent means that the request was processed.
        // In this case, the function was deleted, and the return value
        // is intentionally blank.
        return response.HttpStatusCode == System.Net.HttpStatusCode.NoContent;
    }
```

- API の詳細については、「 API リファレンスDeleteFunction」の「」を参照してください。AWS SDK for .NET

`GetFunction`

次の例は、`GetFunction` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
/// Gets information about a Lambda function.
/// </summary>
/// <param name="functionName">The name of the Lambda function for
/// which to retrieve information.</param>
/// <returns>Async Task.</returns>
public async Task<FunctionConfiguration> GetFunctionAsync(string functionName)
{
    var functionRequest = new GetFunctionRequest
    {
        FunctionName = functionName,
    };

    var response = await _lambdaService.GetFunctionAsync(functionRequest);
    return response.Configuration;
}
```

- API の詳細については、「 API リファレンスGetFunction」の「」を参照してください。 AWS SDK for .NET

`Invoke`

次の例は、Invoke を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。用例一覧を検索し、AWS コードサンプル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>
    /// Invoke a Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the Lambda function to
    /// invoke.</param>
    /// <param name="parameters">The parameter values that will be passed to the
function.</param>
    /// <returns>A System Threading Task.</returns>
    public async Task<string> InvokeFunctionAsync(
        string functionName,
        string parameters)
    {
        var payload = parameters;
        var request = new InvokeRequest
        {
            FunctionName = functionName,
            Payload = payload,
        };

        var response = await _lambdaService.InvokeAsync(request);
        MemoryStream stream = response.Payload;
        string returnValue = System.Text.Encoding.UTF8.GetString(stream.ToArray());
        return returnValue;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「Invoke」を参照してください。

`ListFunctions`

次の例は、`ListFunctions` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get a list of Lambda functions.
    /// </summary>
    /// <returns>A list of FunctionConfiguration objects.</returns>
    public async Task<List<FunctionConfiguration>> ListFunctionsAsync()
    {
        var functionList = new List<FunctionConfiguration>();

        var functionPaginator =
            _lambdaService.Paginators.ListFunctions(new ListFunctionsRequest());
        await foreach (var function in functionPaginator.Functions)
        {
            functionList.Add(function);
        }

        return functionList;
    }
```

- API の詳細については、「 API リファレンスListFunctions」の「」を参照してください。 AWS SDK for .NET

`UpdateFunctionCode`

次の例は、UpdateFunctionCode を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Update an existing Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the Lambda function to update.</
param>
    /// <param name="bucketName">The bucket where the zip file containing
```

```
    /// the Lambda function code is stored.</param>
    /// <param name="key">The key name of the source code file.</param>
    /// <returns>Async Task.</returns>
    public async Task UpdateFunctionCodeAsync(
        string functionName,
        string bucketName,
        string key)
    {
        var functionCodeRequest = new UpdateFunctionCodeRequest
        {
            FunctionName = functionName,
            Publish = true,
            S3Bucket = bucketName,
            S3Key = key,
        };

        var response = await
_lambdaService.UpdateFunctionCodeAsync(functionCodeRequest);
        Console.WriteLine($"The Function was last modified at
{response.LastModified}.");
    }
```

- API の詳細については、「 API リファレンスUpdateFunctionCode」の「」を参照してください。 AWS SDK for .NET

`UpdateFunctionConfiguration`

次の例は、`UpdateFunctionConfiguration` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Update the code of a Lambda function.
```

```
    /// </summary>
    /// <param name="functionName">The name of the function to update.</param>
    /// <param name="functionHandler">The code that performs the function's
 actions.</param>
    /// <param name="environmentVariables">A dictionary of environment variables.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> UpdateFunctionConfigurationAsync(
        string functionName,
        string functionHandler,
        Dictionary<string, string> environmentVariables)
    {
        var request = new UpdateFunctionConfigurationRequest
        {
            Handler = functionHandler,
            FunctionName = functionName,
            Environment = new Amazon.Lambda.Model.Environment { Variables =
 environmentVariables },
        };

        var response = await
 _lambdaService.UpdateFunctionConfigurationAsync(request);

        Console.WriteLine(response.LastModified);

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスUpdateFunctionConfiguration」の「」を参照してください。 AWS SDK for .NET

シナリオ

関数の使用を開始します

次のコードサンプルは、以下の操作方法を示しています。

- IAM ロールと Lambda 関数を作成し、ハンドラーコードをアップロードします。
- 1 つのパラメーターで関数を呼び出して、結果を取得します。
- 関数コードを更新し、環境変数で設定します。

- 新しいパラメーターで関数を呼び出して、結果を取得します。返された実行ログを表示します。
- アカウントの関数を一覧表示し、リソースをクリーンアップします。

詳細については、「コンソールで Lambda 関数を作成する」を参照してください。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Lambda アクションを実行するメソッドを作成します。

```
namespace LambdaActions;

using Amazon.Lambda;
using Amazon.Lambda.Model;

/// <summary>
/// A class that implements AWS Lambda methods.
/// </summary>
public class LambdaWrapper
{
    private readonly IAmazonLambda _lambdaService;

    /// <summary>
    /// Constructor for the LambdaWrapper class.
    /// </summary>
    /// <param name="lambdaService">An initialized Lambda service client.</param>
    public LambdaWrapper(IAmazonLambda lambdaService)
    {
        _lambdaService = lambdaService;
    }

    /// <summary>
    /// Creates a new Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the function.</param>
    /// <param name="s3Bucket">The Amazon Simple Storage Service (Amazon S3)
```

```
    /// bucket where the zip file containing the code is located.</param>
    /// <param name="s3Key">The Amazon S3 key of the zip file.</param>
    /// <param name="role">The Amazon Resource Name (ARN) of a role with the
    /// appropriate Lambda permissions.</param>
    /// <param name="handler">The name of the handler function.</param>
    /// <returns>The Amazon Resource Name (ARN) of the newly created
    /// Lambda function.</returns>
    public async Task<string> CreateLambdaFunctionAsync(
        string functionName,
        string s3Bucket,
        string s3Key,
        string role,
        string handler)
    {
        // Defines the location for the function code.
        // S3Bucket - The S3 bucket where the file containing
        //            the source code is stored.
        // S3Key    - The name of the file containing the code.
        var functionCode = new FunctionCode
        {
            S3Bucket = s3Bucket,
            S3Key = s3Key,
        };

        var createFunctionRequest = new CreateFunctionRequest
        {
            FunctionName = functionName,
            Description = "Created by the Lambda .NET API",
            Code = functionCode,
            Handler = handler,
            Runtime = Runtime.Dotnet6,
            Role = role,
        };

        var reponse = await
_lambdaService.CreateFunctionAsync(createFunctionRequest);
        return reponse.FunctionArn;
    }


    /// <summary>
    /// Delete an AWS Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the Lambda function to
```

```
    /// delete.</param>
    /// <returns>A Boolean value that indicates the success of the action.</returns>
    public async Task<bool> DeleteFunctionAsync(string functionName)
    {
        var request = new DeleteFunctionRequest
        {
            FunctionName = functionName,
        };

        var response = await _lambdaService.DeleteFunctionAsync(request);

        // A return value of NoContent means that the request was processed.
        // In this case, the function was deleted, and the return value
        // is intentionally blank.
        return response.HttpStatusCode == System.Net.HttpStatusCode.NoContent;
    }


    /// <summary>
    /// Gets information about a Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the Lambda function for
    /// which to retrieve information.</param>
    /// <returns>Async Task.</returns>
    public async Task<FunctionConfiguration> GetFunctionAsync(string functionName)
    {
        var functionRequest = new GetFunctionRequest
        {
            FunctionName = functionName,
        };

        var response = await _lambdaService.GetFunctionAsync(functionRequest);
        return response.Configuration;
    }


    /// <summary>
    /// Invoke a Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the Lambda function to
    /// invoke.</param
    /// <param name="parameters">The parameter values that will be passed to the
function.</param>
    /// <returns>A System Threading Task.</returns>
```

```
    public async Task<string> InvokeFunctionAsync(
        string functionName,
        string parameters)
    {
        var payload = parameters;
        var request = new InvokeRequest
        {
            FunctionName = functionName,
            Payload = payload,
        };

        var response = await _lambdaService.InvokeAsync(request);
        MemoryStream stream = response.Payload;
        string returnValue = System.Text.Encoding.UTF8.GetString(stream.ToArray());
        return returnValue;
    }


    /// <summary>
    /// Get a list of Lambda functions.
    /// </summary>
    /// <returns>A list of FunctionConfiguration objects.</returns>
    public async Task<List<FunctionConfiguration>> ListFunctionsAsync()
    {
        var functionList = new List<FunctionConfiguration>();

        var functionPaginator =
            _lambdaService.Paginators.ListFunctions(new ListFunctionsRequest());
        await foreach (var function in functionPaginator.Functions)
        {
            functionList.Add(function);
        }

        return functionList;
    }


    /// <summary>
    /// Update an existing Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the Lambda function to update.</
param>
    /// <param name="bucketName">The bucket where the zip file containing
    /// the Lambda function code is stored.</param>
```

```
    /// <param name="key">The key name of the source code file.</param>
    /// <returns>Async Task.</returns>
    public async Task UpdateFunctionCodeAsync(
        string functionName,
        string bucketName,
        string key)
    {
        var functionCodeRequest = new UpdateFunctionCodeRequest
        {
            FunctionName = functionName,
            Publish = true,
            S3Bucket = bucketName,
            S3Key = key,
        };

        var response = await
 _lambdaService.UpdateFunctionCodeAsync(functionCodeRequest);
        Console.WriteLine($"The Function was last modified at
 {response.LastModified}.");
    }


    /// <summary>
    /// Update the code of a Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the function to update.</param>
    /// <param name="functionHandler">The code that performs the function's
 actions.</param>
    /// <param name="environmentVariables">A dictionary of environment variables.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> UpdateFunctionConfigurationAsync(
        string functionName,
        string functionHandler,
        Dictionary<string, string> environmentVariables)
    {
        var request = new UpdateFunctionConfigurationRequest
        {
            Handler = functionHandler,
            FunctionName = functionName,
            Environment = new Amazon.Lambda.Model.Environment { Variables =
 environmentVariables },
        };
```

```
        var response = await
 _lambdaService.UpdateFunctionConfigurationAsync(request);

        Console.WriteLine(response.LastModified);

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


}
```

シナリオを実行する関数を作成します。

```
global using System.Threading.Tasks;
global using Amazon.IdentityManagement;
global using Amazon.Lambda;
global using LambdaActions;
global using LambdaScenarioCommon;
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.Lambda.Model;
using Microsoft.Extensions.Configuration;

namespace LambdaBasics;

public class LambdaBasics
{
    private static ILogger logger = 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<IAmazonLambda>()
            .AddAWSService<IAmazonIdentityManagementService>()
            .AddTransient<LambdaWrapper>()
            .AddTransient<LambdaRoleWrapper>()
            .AddTransient<UIWrapper>()
        )
        .Build();

        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();


        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<LambdaBasics>();

        var lambdaWrapper = host.Services.GetRequiredService<LambdaWrapper>();
        var lambdaRoleWrapper =
host.Services.GetRequiredService<LambdaRoleWrapper>();
        var uiWrapper = host.Services.GetRequiredService<UIWrapper>();

        string functionName = configuration["FunctionName"]!;
        string roleName = configuration["RoleName"]!;
        string policyDocument = "{" +
            " \"Version\": \"2012-10-17\"," +
            " \"Statement\": [ " +
            "    {" +
            "        \"Effect\": \"Allow\"," +
            "        \"Principal\": {" +
            "            \"Service\": \"lambda.amazonaws.com\" " +
            "    }," +
            "        \"Action\": \"sts:AssumeRole\" " +
            "    }" +
            "]" +
        "}";

        var incrementHandler = configuration["IncrementHandler"];
```

```
        var calculatorHandler = configuration["CalculatorHandler"];
        var bucketName = configuration["BucketName"];
        var incrementKey = configuration["IncrementKey"];
        var calculatorKey = configuration["CalculatorKey"];
        var policyArn = configuration["PolicyArn"];

        uiWrapper.DisplayLambdaBasicsOverview();

        // Create the policy to use with the AWS Lambda functions and then attach
the
        // policy to a new role.
        var roleArn = await lambdaRoleWrapper.CreateLambdaRoleAsync(roleName,
policyDocument);

        Console.WriteLine("Waiting for role to become active.");
        uiWrapper.WaitABit(15, "Wait until the role is active before trying to use
it.");

        // Attach the appropriate AWS Identity and Access Management (IAM) role
policy to the new role.
        var success = await lambdaRoleWrapper.AttachLambdaRolePolicyAsync(policyArn,
roleName);
        uiWrapper.WaitABit(10, "Allow time for the IAM policy to be attached to the
role.");

        // Create the Lambda function using a zip file stored in an Amazon Simple
Storage Service
        // (Amazon S3) bucket.
        uiWrapper.DisplayTitle("Create Lambda Function");
        Console.WriteLine($"Creating the AWS Lambda function: {functionName}.");
        var lambdaArn = await lambdaWrapper.CreateLambdaFunctionAsync(
            functionName,
            bucketName,
            incrementKey,
            roleArn,
            incrementHandler);

        Console.WriteLine("Waiting for the new function to be available.");
        Console.WriteLine($"The AWS Lambda ARN is {lambdaArn}");

        // Get the Lambda function.
        Console.WriteLine($"Getting the {functionName} AWS Lambda function.");
        FunctionConfiguration config;
        do
```

```
        {
            config = await lambdaWrapper.GetFunctionAsync(functionName);
            Console.Write(".");
        }
        while (config.State != State.Active);

        Console.WriteLine($"\nThe function, {functionName} has been created.");
        Console.WriteLine($"The runtime of this Lambda function is
{config.Runtime}.");

        uiWrapper.PressEnter();

        // List the Lambda functions.
        uiWrapper.DisplayTitle("Listing all Lambda functions.");
        var functions = await lambdaWrapper.ListFunctionsAsync();
        DisplayFunctionList(functions);

        uiWrapper.DisplayTitle("Invoke increment function");
        Console.WriteLine("Now that it has been created, invoke the Lambda increment
function.");
        string? value;
        do
        {
            Console.Write("Enter a value to increment: ");
            value = Console.ReadLine();
        }
        while (string.IsNullOrEmpty(value));

        string functionParameters = "{" +
            "\"action\": \"increment\", " +
            "\"x\": \"" + value + "\"" +
        "}";
        var answer = await lambdaWrapper.InvokeFunctionAsync(functionName,
functionParameters);
        Console.WriteLine($"{value} + 1 = {answer}.");

        uiWrapper.DisplayTitle("Update function");
        Console.WriteLine("Now update the Lambda function code.");
        await lambdaWrapper.UpdateFunctionCodeAsync(functionName, bucketName,
calculatorKey);

        do
        {
            config = await lambdaWrapper.GetFunctionAsync(functionName);
```

```
            Console.Write(".");
        }
        while (config.LastUpdateStatus == LastUpdateStatus.InProgress);

        await lambdaWrapper.UpdateFunctionConfigurationAsync(
            functionName,
            calculatorHandler,
            new Dictionary<string, string> { { "LOG_LEVEL", "DEBUG" } });

        do
        {
            config = await lambdaWrapper.GetFunctionAsync(functionName);
            Console.Write(".");
        }
        while (config.LastUpdateStatus == LastUpdateStatus.InProgress);

        uiWrapper.DisplayTitle("Call updated function");
        Console.WriteLine("Now call the updated function...");

        bool done = false;

        do
        {
            string? opSelected;

            Console.WriteLine("Select the operation to perform:");
            Console.WriteLine("\t1. add");
            Console.WriteLine("\t2. subtract");
            Console.WriteLine("\t3. multiply");
            Console.WriteLine("\t4. divide");
            Console.WriteLine("\tOr enter \"q\" to quit.");
            Console.WriteLine("Enter the number (1, 2, 3, 4, or q) of the operation
you want to perform: ");
            do
            {
                Console.Write("Your choice? ");
                opSelected = Console.ReadLine();
            }
            while (opSelected == string.Empty);

            var operation = (opSelected) switch
            {
                "1" => "add",
                "2" => "subtract",
```

```
                "3" => "multiply",
                "4" => "divide",
                "q" => "quit",
                _ => "add",
            };

            if (operation == "quit")
            {
                done = true;
            }
            else
            {
                // Get two numbers and an action from the user.
                value = string.Empty;
                do
                {
                    Console.Write("Enter the first value: ");
                    value = Console.ReadLine();
                }
                while (value == string.Empty);

                string? value2;
                do
                {
                    Console.Write("Enter a second value: ");
                    value2 = Console.ReadLine();
                }
                while (value2 == string.Empty);

                functionParameters = "{" +
                    "\"action\": \"" + operation + "\", " +
                    "\"x\": \"" + value + "\"," +
                    "\"y\": \"" + value2 + "\"" +
                "}";

                answer = await lambdaWrapper.InvokeFunctionAsync(functionName,
functionParameters);
                Console.WriteLine($"The answer when we {operation} the two numbers
is: {answer}.");
            }

            uiWrapper.PressEnter();
        } while (!done);
```

```
        // Delete the function created earlier.

        uiWrapper.DisplayTitle("Clean up resources");
        // Detach the IAM policy from the IAM role.
        Console.WriteLine("First detach the IAM policy from the role.");
        success = await lambdaRoleWrapper.DetachLambdaRolePolicyAsync(policyArn,
roleName);
        uiWrapper.WaitABit(15, "Let's wait for the policy to be fully detached from
the role.");

        Console.WriteLine("Delete the AWS Lambda function.");
        success = await lambdaWrapper.DeleteFunctionAsync(functionName);
        if (success)
        {
            Console.WriteLine($"The {functionName} function was deleted.");
        }
        else
        {
            Console.WriteLine($"Could not remove the function {functionName}");
        }

        // Now delete the IAM role created for use with the functions
        // created by the application.
        Console.WriteLine("Now we can delete the role that we created.");
        success = await lambdaRoleWrapper.DeleteLambdaRoleAsync(roleName);
        if (success)
        {
            Console.WriteLine("The role has been successfully removed.");
        }
        else
        {
            Console.WriteLine("Couldn't delete the role.");
        }

        Console.WriteLine("The Lambda Scenario is now complete.");
        uiWrapper.PressEnter();

        // Displays a formatted list of existing functions returned by the
        // LambdaMethods.ListFunctions.
        void DisplayFunctionList(List<FunctionConfiguration> functions)
        {
            functions.ForEach(functionConfig =>
            {
```

```
 Console.WriteLine($"{functionConfig.FunctionName}\t{functionConfig.Description}");
            });
        }
    }
}


namespace LambdaActions;

using Amazon.IdentityManagement;
using Amazon.IdentityManagement.Model;

public class LambdaRoleWrapper
{
    private readonly IAmazonIdentityManagementService _lambdaRoleService;

    public LambdaRoleWrapper(IAmazonIdentityManagementService lambdaRoleService)
    {
        _lambdaRoleService = lambdaRoleService;
    }

    /// <summary>
    /// Attach an AWS Identity and Access Management (IAM) role policy to the
    /// IAM role to be assumed by the AWS Lambda functions created for the scenario.
    /// </summary>
    /// <param name="policyArn">The Amazon Resource Name (ARN) of the IAM policy.</
param>
    /// <param name="roleName">The name of the IAM role to attach the IAM policy
 to.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AttachLambdaRolePolicyAsync(string policyArn, string
 roleName)
    {
        var response = await _lambdaRoleService.AttachRolePolicyAsync(new
 AttachRolePolicyRequest { PolicyArn = policyArn, RoleName = roleName });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Create a new IAM role.
    /// </summary>
    /// <param name="roleName">The name of the IAM role to create.</param>
```

```
    /// <param name="policyDocument">The policy document for the new IAM role.</
param>
    /// <returns>A string representing the ARN for newly created role.</returns>
    public async Task<string> CreateLambdaRoleAsync(string roleName, string
 policyDocument)
    {
        var request = new CreateRoleRequest
        {
            AssumeRolePolicyDocument = policyDocument,
            RoleName = roleName,
        };

        var response = await _lambdaRoleService.CreateRoleAsync(request);
        return response.Role.Arn;
    }

    /// <summary>
    /// Deletes an IAM role.
    /// </summary>
    /// <param name="roleName">The name of the role to delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</returns>
    public async Task<bool> DeleteLambdaRoleAsync(string roleName)
    {
        var request = new DeleteRoleRequest
        {
            RoleName = roleName,
        };

        var response = await _lambdaRoleService.DeleteRoleAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    public async Task<bool> DetachLambdaRolePolicyAsync(string policyArn, string
 roleName)
    {
        var response = await _lambdaRoleService.DetachRolePolicyAsync(new
 DetachRolePolicyRequest { PolicyArn = policyArn, RoleName = roleName });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
}


namespace LambdaScenarioCommon;
public class UIWrapper
```

```
{
    public readonly string SepBar = new('-', Console.WindowWidth);

    /// <summary>
    /// Show information about the AWS Lambda Basics scenario.
    /// </summary>
    public void DisplayLambdaBasicsOverview()
    {
        Console.Clear();

        DisplayTitle("Welcome to AWS Lambda Basics");
        Console.WriteLine("This example application does the following:");
        Console.WriteLine("\t1. Creates an AWS Identity and Access Management (IAM)
 role that will be assumed by the functions we create.");
        Console.WriteLine("\t2. Attaches an IAM role policy that has Lambda
 permissions.");
        Console.WriteLine("\t3. Creates a Lambda function that increments the value
 passed to it.");
        Console.WriteLine("\t4. Calls the increment function and passes a value.");
        Console.WriteLine("\t5. Updates the code so that the function is a simple
 calculator.");
        Console.WriteLine("\t6. Calls the calculator function with the values
 entered.");
        Console.WriteLine("\t7. Deletes the Lambda function.");
        Console.WriteLine("\t7. Detaches the IAM role policy.");
        Console.WriteLine("\t8. Deletes the IAM role.");
        PressEnter();
    }

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

数値をインクリメントする Lambda ハンドラーを定義します。

```
using Amazon.Lambda.Core;
```

```
// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace LambdaIncrement;

public class Function
{

    /// <summary>
    /// A simple function increments the integer parameter.
    /// </summary>
    /// <param name="input">A JSON string containing an action, which must be
    /// "increment" and a string representing the value to increment.</param>
    /// <param name="context">The context object passed by Lambda containing
    /// information about invocation, function, and execution environment.</param>
    /// <returns>A string representing the incremented value of the parameter.</
returns>
    public int FunctionHandler(Dictionary<string, string> input, ILambdaContext
 context)
    {
        if (input["action"] == "increment")
        {
            int inputValue = Convert.ToInt32(input["x"]);
            return inputValue + 1;
        }
        else
        {
            return 0;
        }
    }
}
```

算術演算を実行する 2 番目の Lambda ハンドラーを定義します。

```
using Amazon.Lambda.Core;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
```

```
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace LambdaCalculator;

public class Function
{

    /// <summary>
    /// A simple function that takes two number in string format and performs
    /// the requested arithmetic function.
    /// </summary>
    /// <param name="input">JSON data containing an action, and x and y values.
    /// Valid actions include: add, subtract, multiply, and divide.</param>
    /// <param name="context">The context object passed by Lambda containing
    /// information about invocation, function, and execution environment.</param>
    /// <returns>A string representing the results of the calculation.</returns>
    public int FunctionHandler(Dictionary<string, string> input, ILambdaContext
 context)
    {
        var action = input["action"];
        int x = Convert.ToInt32(input["x"]);
        int y = Convert.ToInt32(input["y"]);
        int result;
        switch (action)
        {
            case "add":
                result = x + y;
                break;
            case "subtract":
                result = x - y;
                break;
            case "multiply":
                result = x * y;
                break;
            case "divide":
                if (y == 0)
                {
                    Console.Error.WriteLine("Divide by zero error.");
                    result = 0;
                }
                else
                    result = x / y;
                break;
```

```
            default:
                Console.Error.WriteLine($"{action} is not a valid operation.");
                result = 0;
                break;
        }
        return result;
    }
}
```

- 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。
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

サーバーレスサンプル

Kinesis トリガーから Lambda 関数を呼び出す

次のコード例では、Kinesis ストリームからレコードを受信することによってトリガーされるイベントを受け取る、Lambda 関数の実装方法を示しています。この関数は Kinesis ペイロードを取得し、それを Base64 からデコードして、そのレコードの内容をログ記録します。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

.NET を使用した Lambda での Kinesis イベントの消費。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace KinesisIntegrationSampleCode;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task FunctionHandler(KinesisEvent evnt, ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return;
        }

        foreach (var record in evnt.Records)
        {
            try
            {
                Logger.LogInformation($"Processed Event with EventId:
 {record.EventId}");
                string data = await GetRecordDataAsync(record.Kinesis, context);
                Logger.LogInformation($"Data: {data}");
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex)
            {
                Logger.LogError($"An error occurred {ex.Message}");
                throw;
            }
        }
```

```
        Logger.LogInformation($"Successfully processed {evnt.Records.Count}
 records.");
    }

    private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
 ILambdaContext context)
    {
        byte[] bytes = record.Data.ToArray();
        string data = Encoding.UTF8.GetString(bytes);
        await Task.CompletedTask; //Placeholder for actual async work
        return data;
    }
}
```

DynamoDB トリガーから Lambda 関数を呼び出す

次のコード例は、DynamoDB ストリームからレコードを受信することによってトリガーされるイベントを受信する Lambda 関数を実装する方法を示しています。関数は DynamoDB ペイロードを取得し、レコードの内容をログ記録します。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

.NET を使用して Lambda で DynamoDB イベントの消費。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali
```

```
namespace AWSLambda_DDB;

public class Function
{
    public void FunctionHandler(DynamoDBEvent dynamoEvent, ILambdaContext context)
    {
        context.Logger.LogInformation($"Beginning to process
 {dynamoEvent.Records.Count} records...");

        foreach (var record in dynamoEvent.Records)
        {
            context.Logger.LogInformation($"Event ID: {record.EventID}");
            context.Logger.LogInformation($"Event Name: {record.EventName}");

            context.Logger.LogInformation(JsonSerializer.Serialize(record));
        }

        context.Logger.LogInformation("Stream processing complete.");
    }
}
```

Amazon S3 トリガーから Lambda 関数を呼び出す

次のコード例は、S3 バケットにオブジェクトをアップロードすることによってトリガーされるイベントを受け取る Lambda 関数を実装する方法を示しています。この関数は、イベントパラメータから S3 バケット名とオブジェクトキーを取得し、Amazon S3 API を呼び出してオブジェクトのコンテンツタイプを取得してログに記録します。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

.NET を使用して Lambda で S3 イベントを消費します。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
// SPDX-License-Identifier: Apache-2.0
using System.Threading.Tasks;
using Amazon.Lambda.Core;
using Amazon.S3;
using System;
using Amazon.Lambda.S3Events;
using System.Web;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace S3Integration
{
    public class Function
    {
        private static AmazonS3Client _s3Client;
        public Function() : this(null)
        {
        }

        internal Function(AmazonS3Client s3Client)
        {
            _s3Client = s3Client ?? new AmazonS3Client();
        }

        public async Task<string> Handler(S3Event evt, ILambdaContext context)
        {
            try
            {
                if (evt.Records.Count <= 0)
                {
                    context.Logger.LogLine("Empty S3 Event received");
                    return string.Empty;
                }

                var bucket = evt.Records[0].S3.Bucket.Name;
                var key = HttpUtility.UrlDecode(evt.Records[0].S3.Object.Key);

                context.Logger.LogLine($"Request is for {bucket} and {key}");

                var objectResult = await _s3Client.GetObjectAsync(bucket, key);
```

```
                context.Logger.LogLine($"Returning {objectResult.Key}");

                return objectResult.Key;
            }
            catch (Exception e)
            {
                context.Logger.LogLine($"Error processing request - {e.Message}");

                return string.Empty;
            }
        }
    }
}
```

Amazon SNS トリガーから Lambda 関数を呼び出す

次のコード例は、SNS トピックからメッセージを受信することによってトリガーされるイベントを受け取る Lambda 関数を実装する方法を示しています。この関数はイベントパラメータからメッセージを取得し、各メッセージの内容を記録します。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行する方法を確認してください。

.NET を使用した Lambda での SNS イベントの消費。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SNSEvents;


// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali
```

```
namespace SnsIntegration;

public class Function
{
    public async Task FunctionHandler(SNSEvent evnt, ILambdaContext context)
    {
        foreach (var record in evnt.Records)
        {
            await ProcessRecordAsync(record, context);
        }
        context.Logger.LogInformation("done");
    }

    private async Task ProcessRecordAsync(SNSEvent.SNSRecord record, ILambdaContext
 context)
    {
        try
        {
            context.Logger.LogInformation($"Processed record {record.Sns.Message}");

            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
        catch (Exception e)
        {
            //You can use Dead Letter Queue to handle failures. By configuring a
 Lambda DLQ.
            context.Logger.LogError($"An error occurred");
            throw;
        }
    }
}
```

Amazon SQS トリガーから Lambda 関数を呼び出す

次のコード例では、SQS キューからメッセージを受信することによってトリガーされるイベントを受け取る、Lambda 関数の実装方法を示しています。この関数はイベントパラメータからメッセージを取得し、各メッセージの内容を記録します。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

.NET を使用した Lambda での SQS イベントの消費。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;


// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace SqsIntegrationSampleCode
{
    public async Task FunctionHandler(SQSEvent evnt, ILambdaContext context)
    {
        foreach (var message in evnt.Records)
        {
            await ProcessMessageAsync(message, context);
        }

        context.Logger.LogInformation("done");
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
 ILambdaContext context)
    {
        try
        {
            context.Logger.LogInformation($"Processed message {message.Body}");

            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
```

```
        catch (Exception e)
        {
            //You can use Dead Letter Queue to handle failures. By configuring a
 Lambda DLQ.
            context.Logger.LogError($"An error occurred");
            throw;
        }

    }
}
```

Kinesis トリガーを使用した Lambda 関数でのバッチアイテムの失敗のレポート

以下のコード例では、Kinesis ストリームからイベントを受け取る Lambda 関数のための、部分的なバッチレスポンスの実装方法を示しています。この関数は、レスポンスとしてバッチアイテムの失敗を報告し、対象のメッセージを後で再試行するよう Lambda に伝えます。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

.NET を使用した Lambda での Kinesis バッチアイテム失敗のレポート。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using System.Text.Json.Serialization;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali
```

```
namespace KinesisIntegration;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task<StreamsEventResponse> FunctionHandler(KinesisEvent evnt,
 ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return new StreamsEventResponse();
        }

        foreach (var record in evnt.Records)
        {
            try
            {
                Logger.LogInformation($"Processed Event with EventId:
 {record.EventId}");
                string data = await GetRecordDataAsync(record.Kinesis, context);
                Logger.LogInformation($"Data: {data}");
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex)
            {
                Logger.LogError($"An error occurred {ex.Message}");
                /* Since we are working with streams, we can return the failed item
 immediately.
                   Lambda will immediately begin to retry processing from this
 failed item onwards. */
                return new StreamsEventResponse
                {
                    BatchItemFailures = new
 List<StreamsEventResponse.BatchItemFailure>
                    {
                        new StreamsEventResponse.BatchItemFailure { ItemIdentifier =
 record.Kinesis.SequenceNumber }
                    }
                };
            }
        }
```

```
        Logger.LogInformation($"Successfully processed {evnt.Records.Count}
 records.");
        return new StreamsEventResponse();
    }

    private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
 ILambdaContext context)
    {
        byte[] bytes = record.Data.ToArray();
        string data = Encoding.UTF8.GetString(bytes);
        await Task.CompletedTask; //Placeholder for actual async work
        return data;
    }
}

public class StreamsEventResponse
{
    [JsonPropertyName("batchItemFailures")]
    public IList<BatchItemFailure> BatchItemFailures { get; set; }
    public class BatchItemFailure
    {
        [JsonPropertyName("itemIdentifier")]
        public string ItemIdentifier { get; set; }
    }
}
```

DynamoDB トリガーで Lambda 関数のバッチアイテムの失敗をレポートする

次のコード例は、DynamoDB ストリームからイベントを受信する Lambda 関数に部分的なバッチレスポンスを実装する方法を示しています。この関数は、レスポンスとしてバッチアイテムの失敗を報告し、対象のメッセージを後で再試行するよう Lambda に伝えます。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

.NET を使用して Lambda で DynamoDB のバッチアイテム失敗のレポート。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace AWSLambda_DDB;

public class Function
{
    public StreamsEventResponse FunctionHandler(DynamoDBEvent dynamoEvent,
 ILambdaContext context)

    {
        context.Logger.LogInformation($"Beginning to process
 {dynamoEvent.Records.Count} records...");
        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
 List<StreamsEventResponse.BatchItemFailure>();
        StreamsEventResponse streamsEventResponse = new StreamsEventResponse();

        foreach (var record in dynamoEvent.Records)
        {
            try
            {
                var sequenceNumber = record.Dynamodb.SequenceNumber;
                context.Logger.LogInformation(sequenceNumber);
            }
            catch (Exception ex)
            {
                context.Logger.LogError(ex.Message);
                batchItemFailures.Add(new StreamsEventResponse.BatchItemFailure()
 { ItemIdentifier = record.Dynamodb.SequenceNumber });
            }
        }

        if (batchItemFailures.Count > 0)
        {
```

```
            streamsEventResponse.BatchItemFailures = batchItemFailures;
        }

        context.Logger.LogInformation("Stream processing complete.");
        return streamsEventResponse;
    }
}
```

Amazon SQS トリガーを使用した Lambda 関数でのバッチアイテムの失敗のレポート

以下のコード例では、SQS キューからイベントを受け取る Lambda 関数のための、部分的なバッチレスポンスの実装方法を示しています。この関数は、レスポンスとしてバッチアイテムの失敗を報告し、対象のメッセージを後で再試行するよう Lambda に伝えます。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

.NET を使用した Lambda での SQS バッチアイテム失敗のレポート。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali
namespace sqsSample;

public class Function
{
    public async Task<SQSBatchResponse> FunctionHandler(SQSEvent evnt,
 ILambdaContext context)
    {
```

```
        List<SQSBatchResponse.BatchItemFailure> batchItemFailures = new
 List<SQSBatchResponse.BatchItemFailure>();
        foreach(var message in evnt.Records)
        {
            try
            {
                //process your message
                await ProcessMessageAsync(message, context);
            }
            catch (System.Exception)
            {
                //Add failed message identifier to the batchItemFailures list
                batchItemFailures.Add(new
 SQSBatchResponse.BatchItemFailure{ItemIdentifier=message.MessageId});
            }
        }
        return new SQSBatchResponse(batchItemFailures);
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
 ILambdaContext context)
    {
        if (String.IsNullOrEmpty(message.Body))
        {
            throw new Exception("No Body in SQS Message.");
        }
        context.Logger.LogInformation($"Processed message {message.Body}");
        // TODO: Do interesting work based on the new message
        await Task.CompletedTask;
    }
}
```

MediaConvert を使用した の例 AWS SDK for .NET

次のコード例は、 AWS SDK for .NET で 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 MediaConvert。

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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。このリンクには GitHub、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が記載されています。

開始方法

こんにち MediaConvertは

次のコード例は、 AWS Elemental MediaConvertの使用を開始する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
using Amazon.MediaConvert;
using Amazon.MediaConvert.Model;

namespace MediaConvertActions;

public static class HelloMediaConvert
{
    static async Task Main(string[] args)
    {
        // Create the client using the default profile.
        var mediaConvertClient = new AmazonMediaConvertClient();

        Console.WriteLine($"Hello AWS Elemental MediaConvert! Your MediaConvert Jobs
 are:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get some MediaConvert jobs.
        var response = await mediaConvertClient.ListJobsAsync(
            new ListJobsRequest()
            {
                MaxResults = 10
            }
```

```
            );

        foreach (var job in response.Jobs)
        {
            Console.WriteLine($"\tJob: {job.Id} status {job.Status}");
            Console.WriteLine();
        }
    }
}
```

- API の詳細については、「 API リファレンスDescribeEndpoints」の「」を参照してください。AWS SDK for .NET

トピック

- アクション

アクション

`CreateJob`

次の例は、CreateJob を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ファイルの場所、クライアント、ラッパーを設定します。

```
        // MediaConvert role Amazon Resource Name (ARN).
        // For information on creating this role, see
        // https://docs.aws.amazon.com/mediaconvert/latest/ug/creating-the-iam-role-
in-mediaconvert-configured.html.
        var mediaConvertRole = _configuration["mediaConvertRoleARN"];
```

```
        // Include the file input and output locations in settings.json or
 settings.local.json.
        var fileInput = _configuration["fileInput"];
        var fileOutput = _configuration["fileOutput"];

        AmazonMediaConvertClient mcClient = new AmazonMediaConvertClient();

        var wrapper = new MediaConvertWrapper(mcClient);
```

```
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Creating job for input file {fileInput}.");
        var jobId = await wrapper.CreateJob(mediaConvertRole!, fileInput!,
 fileOutput!);
        Console.WriteLine($"Created job with Job ID: {jobId}");
        Console.WriteLine(new string('-', 80));
```

ラッパーメソッドを使用してジョブを作成し、ジョブ ID を返します。

```
    /// <summary>
    /// Create a job to convert a media file.
    /// </summary>
    /// <param name="mediaConvertRole">The Amazon Resource Name (ARN) of the media
 convert role, as specified here:
    /// https://docs.aws.amazon.com/mediaconvert/latest/ug/creating-the-iam-role-in-
mediaconvert-configured.html</param>
    /// <param name="fileInput">The Amazon Simple Storage Service (Amazon S3)
 location of the input media file.</param>
    /// <param name="fileOutput">The Amazon S3 location for the output media file.</
param>
    /// <returns>The ID of the new job.</returns>
    public async Task<string> CreateJob(string mediaConvertRole, string fileInput,
        string fileOutput)
    {
        CreateJobRequest createJobRequest = new CreateJobRequest
        {
            Role = mediaConvertRole
        };

        createJobRequest.UserMetadata.Add("Customer", "Amazon");
```

```
        JobSettings jobSettings = new JobSettings
        {
            AdAvailOffset = 0,
            TimecodeConfig = new TimecodeConfig
            {
                Source = TimecodeSource.EMBEDDED
            }
        };
        createJobRequest.Settings = jobSettings;

        #region OutputGroup

        OutputGroup ofg = new OutputGroup
        {
            Name = "File Group",
            OutputGroupSettings = new OutputGroupSettings
            {
                Type = OutputGroupType.FILE_GROUP_SETTINGS,
                FileGroupSettings = new FileGroupSettings
                {
                    Destination = fileOutput
                }
            }
        };

        Output output = new Output
        {
            NameModifier = "_1"
        };

        #region VideoDescription

        VideoDescription vdes = new VideoDescription
        {
            ScalingBehavior = ScalingBehavior.DEFAULT,
            TimecodeInsertion = VideoTimecodeInsertion.DISABLED,
            AntiAlias = AntiAlias.ENABLED,
            Sharpness = 50,
            AfdSignaling = AfdSignaling.NONE,
            DropFrameTimecode = DropFrameTimecode.ENABLED,
            RespondToAfd = RespondToAfd.NONE,
            ColorMetadata = ColorMetadata.INSERT,
            CodecSettings = new VideoCodecSettings
            {
```

```
            Codec = VideoCodec.H_264
        }
    };
    output.VideoDescription = vdes;

    H264Settings h264 = new H264Settings
    {
        InterlaceMode = H264InterlaceMode.PROGRESSIVE,
        NumberReferenceFrames = 3,
        Syntax = H264Syntax.DEFAULT,
        Softness = 0,
        GopClosedCadence = 1,
        GopSize = 90,
        Slices = 1,
        GopBReference = H264GopBReference.DISABLED,
        SlowPal = H264SlowPal.DISABLED,
        SpatialAdaptiveQuantization = H264SpatialAdaptiveQuantization.ENABLED,
        TemporalAdaptiveQuantization = H264TemporalAdaptiveQuantization.ENABLED,
        FlickerAdaptiveQuantization = H264FlickerAdaptiveQuantization.DISABLED,
        EntropyEncoding = H264EntropyEncoding.CABAC,
        Bitrate = 5000000,
        FramerateControl = H264FramerateControl.SPECIFIED,
        RateControlMode = H264RateControlMode.CBR,
        CodecProfile = H264CodecProfile.MAIN,
        Telecine = H264Telecine.NONE,
        MinIInterval = 0,
        AdaptiveQuantization = H264AdaptiveQuantization.HIGH,
        CodecLevel = H264CodecLevel.AUTO,
        FieldEncoding = H264FieldEncoding.PAFF,
        SceneChangeDetect = H264SceneChangeDetect.ENABLED,
        QualityTuningLevel = H264QualityTuningLevel.SINGLE_PASS,
        FramerateConversionAlgorithm =
            H264FramerateConversionAlgorithm.DUPLICATE_DROP,
        UnregisteredSeiTimecode = H264UnregisteredSeiTimecode.DISABLED,
        GopSizeUnits = H264GopSizeUnits.FRAMES,
        ParControl = H264ParControl.SPECIFIED,
        NumberBFramesBetweenReferenceFrames = 2,
        RepeatPps = H264RepeatPps.DISABLED,
        FramerateNumerator = 30,
        FramerateDenominator = 1,
        ParNumerator = 1,
        ParDenominator = 1
    };
    output.VideoDescription.CodecSettings.H264Settings = h264;
```

```
        #endregion VideoDescription

        #region AudioDescription

        AudioDescription ades = new AudioDescription
        {
            LanguageCodeControl = AudioLanguageCodeControl.FOLLOW_INPUT,
            // This name matches one specified in the following Inputs.
            AudioSourceName = "Audio Selector 1",
            CodecSettings = new AudioCodecSettings
            {
                Codec = AudioCodec.AAC
            }
        };

        AacSettings aac = new AacSettings
        {
            AudioDescriptionBroadcasterMix =
AacAudioDescriptionBroadcasterMix.NORMAL,
            RateControlMode = AacRateControlMode.CBR,
            CodecProfile = AacCodecProfile.LC,
            CodingMode = AacCodingMode.CODING_MODE_2_0,
            RawFormat = AacRawFormat.NONE,
            SampleRate = 48000,
            Specification = AacSpecification.MPEG4,
            Bitrate = 64000
        };
        ades.CodecSettings.AacSettings = aac;
        output.AudioDescriptions.Add(ades);

        #endregion AudioDescription

        #region Mp4 Container

        output.ContainerSettings = new ContainerSettings
        {
            Container = ContainerType.MP4
        };
        Mp4Settings mp4 = new Mp4Settings
        {
            CslgAtom = Mp4CslgAtom.INCLUDE,
            FreeSpaceBox = Mp4FreeSpaceBox.EXCLUDE,
            MoovPlacement = Mp4MoovPlacement.PROGRESSIVE_DOWNLOAD
```

```
        };
        output.ContainerSettings.Mp4Settings = mp4;

        #endregion Mp4 Container

        ofg.Outputs.Add(output);
        createJobRequest.Settings.OutputGroups.Add(ofg);

        #endregion OutputGroup

        #region Input

        Input input = new Input
        {
            FilterEnable = InputFilterEnable.AUTO,
            PsiControl = InputPsiControl.USE_PSI,
            FilterStrength = 0,
            DeblockFilter = InputDeblockFilter.DISABLED,
            DenoiseFilter = InputDenoiseFilter.DISABLED,
            TimecodeSource = InputTimecodeSource.EMBEDDED,
            FileInput = fileInput
        };

        AudioSelector audsel = new AudioSelector
        {
            Offset = 0,
            DefaultSelection = AudioDefaultSelection.NOT_DEFAULT,
            ProgramSelection = 1,
            SelectorType = AudioSelectorType.TRACK
        };
        audsel.Tracks.Add(1);
        input.AudioSelectors.Add("Audio Selector 1", audsel);

        input.VideoSelector = new VideoSelector
        {
            ColorSpace = ColorSpace.FOLLOW
        };

        createJobRequest.Settings.Inputs.Add(input);

        #endregion Input

        CreateJobResponse createJobResponse =
            await _amazonMediaConvert.CreateJobAsync(createJobRequest);
```

```
        var jobId = createJobResponse.Job.Id;

        return jobId;
    }
```

- API の詳細については、「 API リファレンスCreateJob」の「」を参照してください。 AWS SDK for .NET

`GetJob`

次の例は、GetJob を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ファイルの場所、クライアント、ラッパーを設定します。

```
        // MediaConvert role Amazon Resource Name (ARN).
        // For information on creating this role, see
        // https://docs.aws.amazon.com/mediaconvert/latest/ug/creating-the-iam-role-
in-mediaconvert-configured.html.
        var mediaConvertRole = _configuration["mediaConvertRoleARN"];

        // Include the file input and output locations in settings.json or
 settings.local.json.
        var fileInput = _configuration["fileInput"];
        var fileOutput = _configuration["fileOutput"];

        AmazonMediaConvertClient mcClient = new AmazonMediaConvertClient();

        var wrapper = new MediaConvertWrapper(mcClient);
```

ID でジョブを取得する。

```
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Getting job information for Job ID {jobId}");
        var job = await wrapper.GetJobById(jobId);
        Console.WriteLine($"Job {job.Id} created on {job.CreatedAt:d} has status
{job.Status}.");
        Console.WriteLine(new string('-', 80));
```

```
    /// <summary>
    /// Get the job information for a job by its ID.
    /// </summary>
    /// <param name="jobId">The ID of the job.</param>
    /// <returns>The Job object.</returns>
    public async Task<Job> GetJobById(string jobId)
    {
        var jobResponse = await _amazonMediaConvert.GetJobAsync(
                new GetJobRequest
                {
                    Id = jobId
                });

        return jobResponse.Job;
    }
```

- API の詳細については、「 API リファレンスGetJob」の「」を参照してください。 AWS SDK for .NET

`ListJobs`

次の例は、`ListJobs` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ファイルの場所、クライアント、ラッパーを設定します。

```
        // MediaConvert role Amazon Resource Name (ARN).
        // For information on creating this role, see
        // https://docs.aws.amazon.com/mediaconvert/latest/ug/creating-the-iam-role-
in-mediaconvert-configured.html.
        var mediaConvertRole = _configuration["mediaConvertRoleARN"];

        // Include the file input and output locations in settings.json or
 settings.local.json.
        var fileInput = _configuration["fileInput"];
        var fileOutput = _configuration["fileOutput"];

        AmazonMediaConvertClient mcClient = new AmazonMediaConvertClient();

        var wrapper = new MediaConvertWrapper(mcClient);
```

特定のステータスのジョブを一覧表示します。

```
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Listing all complete jobs.");
        var completeJobs = await wrapper.ListAllJobsByStatus(JobStatus.COMPLETE);
        completeJobs.ForEach(j =>
        {
            Console.WriteLine($"Job {j.Id} created on {j.CreatedAt:d} has status
 {j.Status}.");
        });
```

ページネーターを使用してジョブを一覧表示します。

```
    /// <summary>
    /// List all of the jobs with a particular status using a paginator.
    /// </summary>
    /// <param name="status">The status to use when listing jobs.</param>
    /// <returns>The list of jobs matching the status.</returns>
    public async Task<List<Job>> ListAllJobsByStatus(JobStatus? status = null)
    {
        var returnedJobs = new List<Job>();

        var paginatedJobs = _amazonMediaConvert.Paginators.ListJobs(
```

```
                new ListJobsRequest
                {
                    Status = status
                });

        // Get the entire list using the paginator.
        await foreach (var job in paginatedJobs.Jobs)
        {
            returnedJobs.Add(job);
        }

        return returnedJobs;
    }
```

- API の詳細については、「 API リファレンスListJobs」の「」を参照してください。 AWS SDK for .NET

を使用した Organizations の例 AWS SDK for .NET

次のコード例は、Organizations AWS SDK for .NET で 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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。このリンクには GitHub、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が記載されています。

トピック

- アクション

アクション

`AttachPolicy`

次の例は、`AttachPolicy` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

    /// <summary>
    /// Shows how to attach an AWS Organizations policy to an organization,
    /// an organizational unit, or an account.
    /// </summary>
    public class AttachPolicy
    {
        /// <summary>
        /// Initializes the Organizations client object and then calls the
        /// AttachPolicyAsync method to attach the policy to the root
        /// organization.
        /// </summary>
        public static async Task Main()
        {
            IAmazonOrganizations client = new AmazonOrganizationsClient();
            var policyId = "p-00000000";
            var targetId = "r-0000";

            var request = new AttachPolicyRequest
            {
                PolicyId = policyId,
                TargetId = targetId,
            };

            var response = await client.AttachPolicyAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"Successfully attached Policy ID {policyId} to
Target ID: {targetId}.");
```

```
            }
            else
            {
                Console.WriteLine("Was not successful in attaching the policy.");
            }
        }
    }
```

- API の詳細については、「 API リファレンスAttachPolicy」の「」を参照してください。 AWS SDK for .NET

`CreateAccount`

次の例は、`CreateAccount` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

/// <summary>
/// Creates a new AWS Organizations account.
/// </summary>
public class CreateAccount
{
    /// <summary>
    /// Initializes an Organizations client object and uses it to create
    /// the new account with the name specified in accountName.
    /// </summary>
    public static async Task Main()
    {
```

```
            IAmazonOrganizations client = new AmazonOrganizationsClient();
            var accountName = "ExampleAccount";
            var email = "someone@example.com";

            var request = new CreateAccountRequest
            {
                AccountName = accountName,
                Email = email,
            };

            var response = await client.CreateAccountAsync(request);
            var status = response.CreateAccountStatus;

            Console.WriteLine($"The staus of {status.AccountName} is
{status.State}.");
        }
    }
```

- API の詳細については、「 API リファレンスCreateAccount」の「」を参照してください。AWS SDK for .NET

`CreateOrganization`

次の例は、`CreateOrganization` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

    /// <summary>
```

```
    /// Creates an organization in AWS Organizations.
    /// </summary>
    public class CreateOrganization
    {
        /// <summary>
        /// Creates an Organizations client object and then uses it to create
        /// a new organization with the default user as the administrator, and
        /// then displays information about the new organization.
        /// </summary>
        public static async Task Main()
        {
            IAmazonOrganizations client = new AmazonOrganizationsClient();

            var response = await client.CreateOrganizationAsync(new
CreateOrganizationRequest
            {
                FeatureSet = "ALL",
            });

            Organization newOrg = response.Organization;

            Console.WriteLine($"Organization: {newOrg.Id} Main Accoount:
{newOrg.MasterAccountId}");
        }
    }
```

- API の詳細については、「 API リファレンスCreateOrganization」の「」を参照してください。 AWS SDK for .NET

`CreateOrganizationalUnit`

次の例は、`CreateOrganizationalUnit` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

/// <summary>
/// Creates a new organizational unit in AWS Organizations.
/// </summary>
public class CreateOrganizationalUnit
{
    /// <summary>
    /// Initializes an Organizations client object and then uses it to call
    /// the CreateOrganizationalUnit method. If the call succeeds, it
    /// displays information about the new organizational unit.
    /// </summary>
    public static async Task Main()
    {
        // Create the client object using the default account.
        IAmazonOrganizations client = new AmazonOrganizationsClient();

        var orgUnitName = "ProductDevelopmentUnit";

        var request = new CreateOrganizationalUnitRequest
        {
            Name = orgUnitName,
            ParentId = "r-0000",
        };

        var response = await client.CreateOrganizationalUnitAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"Successfully created organizational unit:
{orgUnitName}.");
            Console.WriteLine($"Organizational unit {orgUnitName} Details");
            Console.WriteLine($"ARN: {response.OrganizationalUnit.Arn} Id:
{response.OrganizationalUnit.Id}");
        }
        else
        {
            Console.WriteLine("Could not create new organizational unit.");
        }
    }
```

```
    }
```

- API の詳細については、「 API リファレンスCreateOrganizationalUnit」の「」を参照してください。 AWS SDK for .NET

`CreatePolicy`

次の例は、`CreatePolicy` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

/// <summary>
/// Creates a new AWS Organizations Policy.
/// </summary>
public class CreatePolicy
{
    /// <summary>
    /// Initializes the AWS Organizations client object, uses it to
    /// create a new Organizations Policy, and then displays information
    /// about the newly created Policy.
    /// </summary>
    public static async Task Main()
    {
        IAmazonOrganizations client = new AmazonOrganizationsClient();
        var policyContent = "{" +
            "   \"Version\": \"2012-10-17\"," +
            " \"Statement\" : [{" +
                " \"Action\" : [\"s3:*\"]," +
```

```
                    " \"Effect\" : \"Allow\"," +
                    " \"Resource\" : \"*\"" +
                "}]" +
            "}";

            try
            {
                var response = await client.CreatePolicyAsync(new
CreatePolicyRequest
                {
                    Content = policyContent,
                    Description = "Enables admins of attached accounts to delegate
all Amazon S3 permissions",
                    Name = "AllowAllS3Actions",
                    Type = "SERVICE_CONTROL_POLICY",
                });

                Policy policy = response.Policy;
                Console.WriteLine($"{policy.PolicySummary.Name} has the following
content: {policy.Content}");
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }
```

- API の詳細については、「 API リファレンスCreatePolicy」の「」を参照してください。 AWS SDK for .NET

DeleteOrganization

次の例は、DeleteOrganization を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

/// <summary>
/// Shows how to delete an existing organization using the AWS
/// Organizations Service.
/// </summary>
public class DeleteOrganization
{
    /// <summary>
    /// Initializes the Organizations client and then calls
    /// DeleteOrganizationAsync to delete the organization.
    /// </summary>
    public static async Task Main()
    {
        // Create the client object using the default account.
        IAmazonOrganizations client = new AmazonOrganizationsClient();

        var response = await client.DeleteOrganizationAsync(new
DeleteOrganizationRequest());

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine("Successfully deleted organization.");
        }
        else
        {
            Console.WriteLine("Could not delete organization.");
        }
    }
}
```

- API の詳細については、「 API リファレンスDeleteOrganization」の「」を参照してください。 AWS SDK for .NET

`DeleteOrganizationalUnit`

次の例は、`DeleteOrganizationalUnit` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

/// <summary>
/// Shows how to delete an existing AWS Organizations organizational unit.
/// </summary>
public class DeleteOrganizationalUnit
{
    /// <summary>
    /// Initializes the Organizations client object and calls
    /// DeleteOrganizationalUnitAsync to delete the organizational unit
    /// with the selected ID.
    /// </summary>
    public static async Task Main()
    {
        // Create the client object using the default account.
        IAmazonOrganizations client = new AmazonOrganizationsClient();

        var orgUnitId = "ou-0000-00000000";

        var request = new DeleteOrganizationalUnitRequest
        {
```

```
                OrganizationalUnitId = orgUnitId,
            };

            var response = await client.DeleteOrganizationalUnitAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"Successfully deleted the organizational unit
with ID: {orgUnitId}.");
            }
            else
            {
                Console.WriteLine($"Could not delete the organizational unit with
ID: {orgUnitId}.");
            }
        }
    }
```

- API の詳細については、「 API リファレンスDeleteOrganizationalUnit」の「」を参照してください。 AWS SDK for .NET

`DeletePolicy`

次の例は、DeletePolicy を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

    /// <summary>
```

```
    /// Deletes an existing AWS Organizations policy.
    /// </summary>
    public class DeletePolicy
    {
        /// <summary>
        /// Initializes the Organizations client object and then uses it to
        /// delete the policy with the specified policyId.
        /// </summary>
        public static async Task Main()
        {
            // Create the client object using the default account.
            IAmazonOrganizations client = new AmazonOrganizationsClient();

            var policyId = "p-00000000";

            var request = new DeletePolicyRequest
            {
                PolicyId = policyId,
            };

            var response = await client.DeletePolicyAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"Successfully deleted Policy: {policyId}.");
            }
            else
            {
                Console.WriteLine($"Could not delete Policy: {policyId}.");
            }
        }
    }
```

- API の詳細については、「 API リファレンスDeletePolicy」の「」を参照してください。 AWS SDK for .NET

DetachPolicy

次の例は、DetachPolicy を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

/// <summary>
/// Shows how to detach a policy from an AWS Organizations organization,
/// organizational unit, or account.
/// </summary>
public class DetachPolicy
{
    /// <summary>
    /// Initializes the Organizations client object and uses it to call
    /// DetachPolicyAsync to detach the policy.
    /// </summary>
    public static async Task Main()
    {
        // Create the client object using the default account.
        IAmazonOrganizations client = new AmazonOrganizationsClient();

        var policyId = "p-00000000";
        var targetId = "r-0000";

        var request = new DetachPolicyRequest
        {
            PolicyId = policyId,
            TargetId = targetId,
        };

        var response = await client.DetachPolicyAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
```

```
                Console.WriteLine($"Successfully detached policy with Policy Id:
{policyId}.");
            }
            else
            {
                Console.WriteLine("Could not detach the policy.");
            }
        }
    }
```

- API の詳細については、「 API リファレンスDetachPolicy」の「」を参照してください。 AWS SDK for .NET

`ListAccounts`

次の例は、`ListAccounts` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

    /// <summary>
    /// Uses the AWS Organizations service to list the accounts associated
    /// with the default account.
    /// </summary>
    public class ListAccounts
    {
        /// <summary>
        /// Creates the Organizations client and then calls its
        /// ListAccountsAsync method.
```

```
        /// </summary>
        public static async Task Main()
        {
            // Create the client object using the default account.
            IAmazonOrganizations client = new AmazonOrganizationsClient();

            var request = new ListAccountsRequest
            {
                MaxResults = 5,
            };

            var response = new ListAccountsResponse();
            try
            {
                do
                {
                    response = await client.ListAccountsAsync(request);
                    response.Accounts.ForEach(a => DisplayAccounts(a));
                    if (response.NextToken is not null)
                    {
                        request.NextToken = response.NextToken;
                    }
                }
                while (response.NextToken is not null);
            }
            catch (AWSOrganizationsNotInUseException ex)
            {
                Console.WriteLine(ex.Message);
            }
        }

        /// <summary>
        /// Displays information about an Organizations account.
        /// </summary>
        /// <param name="account">An Organizations account for which to display
        /// information on the console.</param>
        private static void DisplayAccounts(Account account)
        {
            string accountInfo = $"{account.Id} {account.Name}\t{account.Status}";

            Console.WriteLine(accountInfo);
        }
    }
```

- API の詳細については、「 API リファレンスListAccounts」の「」を参照してください。 AWS SDK for .NET

`ListOrganizationalUnitsForParent`

次の例は、`ListOrganizationalUnitsForParent` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

/// <summary>
/// Lists the AWS Organizations organizational units that belong to an
/// organization.
/// </summary>
public class ListOrganizationalUnitsForParent
{
    /// <summary>
    /// Initializes the Organizations client object and then uses it to
    /// call the ListOrganizationalUnitsForParentAsync method to retrieve
    /// the list of organizational units.
    /// </summary>
    public static async Task Main()
    {
        // Create the client object using the default account.
        IAmazonOrganizations client = new AmazonOrganizationsClient();

        var parentId = "r-0000";

        var request = new ListOrganizationalUnitsForParentRequest
```

```
            {
                ParentId = parentId,
                MaxResults = 5,
            };

            var response = new ListOrganizationalUnitsForParentResponse();
            try
            {
                do
                {
                    response = await
client.ListOrganizationalUnitsForParentAsync(request);
                    response.OrganizationalUnits.ForEach(u =>
DisplayOrganizationalUnit(u));
                    if (response.NextToken is not null)
                    {
                        request.NextToken = response.NextToken;
                    }
                }
                while (response.NextToken is not null);
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }

        /// <summary>
        /// Displays information about an Organizations organizational unit.
        /// </summary>
        /// <param name="unit">The OrganizationalUnit for which to display
        /// information.</param>
        public static void DisplayOrganizationalUnit(OrganizationalUnit unit)
        {
            string accountInfo = $"{unit.Id} {unit.Name}\t{unit.Arn}";

            Console.WriteLine(accountInfo);
        }
    }
```

- API の詳細については、「 API リファレンスListOrganizationalUnitsForParent」の「」を参照してください。 AWS SDK for .NET

`ListPolicies`

次の例は、`ListPolicies` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

/// <summary>
/// Shows how to list the AWS Organizations policies associated with an
/// organization.
/// </summary>
public class ListPolicies
{
    /// <summary>
    /// Initializes an Organizations client object, and then calls its
    /// ListPoliciesAsync method.
    /// </summary>
    public static async Task Main()
    {
        // Create the client object using the default account.
        IAmazonOrganizations client = new AmazonOrganizationsClient();

        // The value for the Filter parameter is required and must must be
        // one of the following:
        //     AISERVICES_OPT_OUT_POLICY
        //     BACKUP_POLICY
        //     SERVICE_CONTROL_POLICY
        //     TAG_POLICY
        var request = new ListPoliciesRequest
```

```
            {
                Filter = "SERVICE_CONTROL_POLICY",
                MaxResults = 5,
            };

            var response = new ListPoliciesResponse();
            try
            {
                do
                {
                    response = await client.ListPoliciesAsync(request);
                    response.Policies.ForEach(p => DisplayPolicies(p));
                    if (response.NextToken is not null)
                    {
                        request.NextToken = response.NextToken;
                    }
                }
                while (response.NextToken is not null);
            }
            catch (AWSOrganizationsNotInUseException ex)
            {
                Console.WriteLine(ex.Message);
            }
        }

        /// <summary>
        /// Displays information about the Organizations policies associated
        /// with an organization.
        /// </summary>
        /// <param name="policy">An Organizations policy summary to display
        /// information on the console.</param>
        private static void DisplayPolicies(PolicySummary policy)
        {
            string policyInfo = $"{policy.Id} {policy.Name}\t{policy.Description}";

            Console.WriteLine(policyInfo);
        }
    }
```

- API の詳細については、「 API リファレンスListPolicies」の「」を参照してください。 AWS SDK for .NET

を使用した Amazon Pinpoint の例 AWS SDK for .NET

次のコード例は、Amazon Pinpoint AWS SDK for .NET で 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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。このリンクには GitHub、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が記載されています。

トピック

- アクション

アクション

`SendMessages`

次の例は、`SendMessages` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

E メールメッセージを送信します。

```
using Amazon;
using Amazon.Pinpoint;
using Amazon.Pinpoint.Model;
using Microsoft.Extensions.Configuration;

namespace SendEmailMessage;
```

```
public class SendEmailMainClass
{
    public static async Task Main(string[] args)
    {
        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();

        // The AWS Region that you want to use to send the email. For a list of
        // AWS Regions where the Amazon Pinpoint API is available, see
        // https://docs.aws.amazon.com/pinpoint/latest/apireference/
        string region = "us-east-1";

        // The "From" address. This address has to be verified in Amazon Pinpoint
        // in the region you're using to send email.
        string senderAddress = configuration["SenderAddress"]!;

        // The address on the "To" line. If your Amazon Pinpoint account is in
        // the sandbox, this address also has to be verified.
        string toAddress = configuration["ToAddress"]!;

        // The Amazon Pinpoint project/application ID to use when you send this
 message.
        // Make sure that the SMS channel is enabled for the project or application
        // that you choose.
        string appId = configuration["AppId"]!;

        try
        {
            await SendEmailMessage(region, appId, toAddress, senderAddress);
        }
        catch (Exception ex)
        {
            Console.WriteLine("The message wasn't sent. Error message: " +
 ex.Message);
        }
    }

    public static async Task<MessageResponse> SendEmailMessage(
        string region, string appId, string toAddress, string senderAddress)
```

```
    {
        var client = new
 AmazonPinpointClient(RegionEndpoint.GetBySystemName(region));

        // The subject line of the email.
        string subject = "Amazon Pinpoint Email test";

        // The body of the email for recipients whose email clients don't
        // support HTML content.
        string textBody = @"Amazon Pinpoint Email Test (.NET)"
                          + "\n---------------------------------"
                          + "\nThis email was sent using the Amazon Pinpoint API
 using the AWS SDK for .NET.";

        // The body of the email for recipients whose email clients support
        // HTML content.
        string htmlBody = @"<html>"
                          + "\n<head></head>"
                          + "\n<body>"
                          + "\n  <h1>Amazon Pinpoint Email Test (AWS SDK for .NET)</
h1>"
                          + "\n  <p>This email was sent using the "
                          + "\n    <a href='https://aws.amazon.com/pinpoint/'>Amazon
 Pinpoint</a> API "
                          + "\n    using the <a href='https://aws.amazon.com/sdk-
for-net/'>AWS SDK for .NET</a>"
                          + "\n  </p>"
                          + "\n</body>"
                          + "\n</html>";

        // The character encoding the you want to use for the subject line and
        // message body of the email.
        string charset = "UTF-8";

        var sendRequest = new SendMessagesRequest
        {
            ApplicationId = appId,
            MessageRequest = new MessageRequest
            {
                Addresses = new Dictionary<string, AddressConfiguration>
                {
                    {
                        toAddress,
                        new AddressConfiguration
```

```
                        {
                            ChannelType = ChannelType.EMAIL
                        }
                    }
                },
                MessageConfiguration = new DirectMessageConfiguration
                {
                    EmailMessage = new EmailMessage
                    {
                        FromAddress = senderAddress,
                        SimpleEmail = new SimpleEmail
                        {
                            HtmlPart = new SimpleEmailPart
                            {
                                Charset = charset,
                                Data = htmlBody
                            },
                            TextPart = new SimpleEmailPart
                            {
                                Charset = charset,
                                Data = textBody
                            },
                            Subject = new SimpleEmailPart
                            {
                                Charset = charset,
                                Data = subject
                            }
                        }
                    }
                }
            }
        };
        Console.WriteLine("Sending message...");
        SendMessagesResponse response = await client.SendMessagesAsync(sendRequest);
        Console.WriteLine("Message sent!");
        return response.MessageResponse;
    }
}
```

SMS メッセージを送信します。

```
using Amazon;
using Amazon.Pinpoint;
using Amazon.Pinpoint.Model;
using Microsoft.Extensions.Configuration;

namespace SendSmsMessage;

public class SendSmsMessageMainClass
{
    public static async Task Main(string[] args)
    {
        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();

        // The AWS Region that you want to use to send the message. For a list of
        // AWS Regions where the Amazon Pinpoint API is available, see
        // https://docs.aws.amazon.com/pinpoint/latest/apireference/
        string region = "us-east-1";

        // The phone number or short code to send the message from. The phone number
        // or short code that you specify has to be associated with your Amazon
 Pinpoint
        // account. For best results, specify long codes in E.164 format.
        string originationNumber = configuration["OriginationNumber"]!;

        // The recipient's phone number.  For best results, you should specify the
        // phone number in E.164 format.
        string destinationNumber = configuration["DestinationNumber"]!;

        // The Pinpoint project/ application ID to use when you send this message.
        // Make sure that the SMS channel is enabled for the project or application
        // that you choose.
        string appId = configuration["AppId"]!;

        // The type of SMS message that you want to send. If you plan to send
        // time-sensitive content, specify TRANSACTIONAL. If you plan to send
        // marketing-related content, specify PROMOTIONAL.
        MessageType messageType = MessageType.TRANSACTIONAL;
```

```
        // The registered keyword associated with the originating short code.
        string? registeredKeyword = configuration["RegisteredKeyword"];

        // The sender ID to use when sending the message. Support for sender ID
        // varies by country or region. For more information, see
        // https://docs.aws.amazon.com/pinpoint/latest/userguide/channels-sms-
countries.html
        string? senderId = configuration["SenderId"];

        try
        {
            var response = await SendSmsMessage(region, appId, destinationNumber,
                originationNumber, registeredKeyword, senderId, messageType);
            Console.WriteLine($"Message sent to
 {response.MessageResponse.Result.Count} recipient(s).");
            foreach (var messageResultValue in
                     response.MessageResponse.Result.Select(r => r.Value))
            {
                Console.WriteLine($"{messageResultValue.MessageId} Status:
 {messageResultValue.DeliveryStatus}");
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine("The message wasn't sent. Error message: " +
 ex.Message);
        }
    }

    public static async Task<SendMessagesResponse> SendSmsMessage(
        string region, string appId, string destinationNumber, string
 originationNumber,
        string? keyword, string? senderId, MessageType messageType)
    {

        // The content of the SMS message.
        string message = "This message was sent through Amazon Pinpoint using" +
                         " the AWS SDK for .NET. Reply STOP to opt out.";


        var client = new
 AmazonPinpointClient(RegionEndpoint.GetBySystemName(region));
```

```
        SendMessagesRequest sendRequest = new SendMessagesRequest
        {
            ApplicationId = appId,
            MessageRequest = new MessageRequest
            {
                Addresses =
                    new Dictionary<string, AddressConfiguration>
                    {
                        {
                            destinationNumber,
                            new AddressConfiguration { ChannelType =
 ChannelType.SMS }
                        }
                    },
                MessageConfiguration = new DirectMessageConfiguration
                {
                    SMSMessage = new SMSMessage
                    {
                        Body = message,
                        MessageType = MessageType.TRANSACTIONAL,
                        OriginationNumber = originationNumber,
                        SenderId = senderId,
                        Keyword = keyword
                    }
                }
            }
        };
        SendMessagesResponse response = await client.SendMessagesAsync(sendRequest);
        return response;
    }
}
```

- API の詳細については、「 API リファレンスSendMessages」の「」を参照してください。 AWS SDK for .NET

を使用した Amazon Polly の例 AWS SDK for .NET

次のコード例は、Amazon Polly AWS SDK for .NET で 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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。このリンクには GitHub、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が記載されています。

トピック

- アクション

アクション

`DeleteLexicon`

次の例は、`DeleteLexicon` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

/// <summary>
/// Deletes an existing Amazon Polly lexicon using the AWS SDK for .NET.
/// </summary>
public class DeleteLexicon
{
    public static async Task Main()
    {
        string lexiconName = "SampleLexicon";
```

```
            var client = new AmazonPollyClient();

            var success = await DeletePollyLexiconAsync(client, lexiconName);

            if (success)
            {
                Console.WriteLine($"Successfully deleted {lexiconName}.");
            }
            else
            {
                Console.WriteLine($"Could not delete {lexiconName}.");
            }
        }

        /// <summary>
        /// Deletes the named Amazon Polly lexicon.
        /// </summary>
        /// <param name="client">The initialized Amazon Polly client object.</param>
        /// <param name="lexiconName">The name of the Amazon Polly lexicon to
        /// delete.</param>
        /// <returns>A Boolean value indicating the success of the operation.</
returns>
        public static async Task<bool> DeletePollyLexiconAsync(
            AmazonPollyClient client,
            string lexiconName)
        {
            var deleteLexiconRequest = new DeleteLexiconRequest()
            {
                Name = lexiconName,
            };

            var response = await client.DeleteLexiconAsync(deleteLexiconRequest);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
    }
```

- API の詳細については、「 API リファレンスDeleteLexicon」の「」を参照してください。 AWS SDK for .NET

DescribeVoices

次の例は、DescribeVoices を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

    public class DescribeVoices
    {
        public static async Task Main()
        {
            var client = new AmazonPollyClient();

            var allVoicesRequest = new DescribeVoicesRequest();
            var enUsVoicesRequest = new DescribeVoicesRequest()
            {
                LanguageCode = "en-US",
            };

            try
            {
                string nextToken;
                do
                {
                    var allVoicesResponse = await
client.DescribeVoicesAsync(allVoicesRequest);
                    nextToken = allVoicesResponse.NextToken;
                    allVoicesRequest.NextToken = nextToken;

                    Console.WriteLine("\nAll voices: ");
                    allVoicesResponse.Voices.ForEach(voice =>
                    {
                        DisplayVoiceInfo(voice);
```

```
                    });
                }
                while (nextToken is not null);

                do
                {
                    var enUsVoicesResponse = await
client.DescribeVoicesAsync(enUsVoicesRequest);
                    nextToken = enUsVoicesResponse.NextToken;
                    enUsVoicesRequest.NextToken = nextToken;

                    Console.WriteLine("\nen-US voices: ");
                    enUsVoicesResponse.Voices.ForEach(voice =>
                    {
                        DisplayVoiceInfo(voice);
                    });
                }
                while (nextToken is not null);
            }
            catch (Exception ex)
            {
                Console.WriteLine("Exception caught: " + ex.Message);
            }
        }

        public static void DisplayVoiceInfo(Voice voice)
        {
            Console.WriteLine($" Name: {voice.Name}\tGender:
{voice.Gender}\tLanguageName: {voice.LanguageName}");
        }
    }
```

- API の詳細については、「 API リファレンスDescribeVoices」の「」を参照してください。AWS SDK for .NET

GetLexicon

次の例は、GetLexicon を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

    /// <summary>
    /// Retrieves information about a specific Amazon Polly lexicon.
    /// </summary>
    public class GetLexicon
    {
        public static async Task Main(string[] args)
        {
            string lexiconName = "SampleLexicon";

            var client = new AmazonPollyClient();

            await GetPollyLexiconAsync(client, lexiconName);
        }

        public static async Task GetPollyLexiconAsync(AmazonPollyClient client,
string lexiconName)
        {
            var getLexiconRequest = new GetLexiconRequest()
            {
                Name = lexiconName,
            };

            try
            {
                var response = await client.GetLexiconAsync(getLexiconRequest);
                Console.WriteLine($"Lexicon:\n Name: {response.Lexicon.Name}");
                Console.WriteLine($"Content: {response.Lexicon.Content}");
            }
            catch (Exception ex)
```

```
            {
                Console.WriteLine("Error: " + ex.Message);
            }
        }
    }
```

- API の詳細については、「 API リファレンスGetLexicon」の「」を参照してください。 AWS SDK for .NET

`ListLexicons`

次の例は、ListLexicons を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

/// <summary>
/// Lists the Amazon Polly lexicons that have been defined. By default,
/// lists the lexicons that are defined in the same AWS Region as the default
/// user. To view Amazon Polly lexicons that are defined in a different AWS
/// Region, supply it as a parameter to the Amazon Polly constructor.
/// </summary>
public class ListLexicons
{
    public static async Task Main()
    {
        var client = new AmazonPollyClient();
        var request = new ListLexiconsRequest();
```

```
            try
            {
                Console.WriteLine("All voices: ");

                do
                {
                    var response = await client.ListLexiconsAsync(request);
                    request.NextToken = response.NextToken;

                    response.Lexicons.ForEach(lexicon =>
                    {
                        var attributes = lexicon.Attributes;
                        Console.WriteLine($"Name: {lexicon.Name}");
                        Console.WriteLine($"\tAlphabet: {attributes.Alphabet}");
                        Console.WriteLine($"\tLanguageCode:
{attributes.LanguageCode}");
                        Console.WriteLine($"\tLastModified:
{attributes.LastModified}");
                        Console.WriteLine($"\tLexemesCount:
{attributes.LexemesCount}");
                        Console.WriteLine($"\tLexiconArn: {attributes.LexiconArn}");
                        Console.WriteLine($"\tSize: {attributes.Size}");
                    });
                }
                while (request.NextToken is not null);
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Error: {ex.Message}");
            }
        }
    }
```

- API の詳細については、「 API リファレンスListLexicons」の「」を参照してください。 AWS SDK for .NET

PutLexicon

次の例は、PutLexicon を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

    /// <summary>
    /// Creates a new Amazon Polly lexicon using the AWS SDK for .NET.
    /// </summary>
    public class PutLexicon
    {
        public static async Task Main()
        {
            string lexiconContent = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>" +
                "<lexicon version=\"1.0\" xmlns=\"http://www.w3.org/2005/01/
pronunciation-lexicon\" xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" " +
                "xsi:schemaLocation=\"http://www.w3.org/2005/01/pronunciation-
lexicon http://www.w3.org/TR/2007/CR-pronunciation-lexicon-20071212/pls.xsd\" " +
                "alphabet=\"ipa\" xml:lang=\"en-US\">" +
                "<lexeme><grapheme>test1</grapheme><alias>test2</alias></lexeme>" +
                "</lexicon>";
            string lexiconName = "SampleLexicon";

            var client = new AmazonPollyClient();
            var putLexiconRequest = new PutLexiconRequest()
            {
                Name = lexiconName,
                Content = lexiconContent,
            };

            try
            {
                var response = await client.PutLexiconAsync(putLexiconRequest);
                if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
                {
```

```
                    Console.WriteLine($"Successfully created Lexicon:
{lexiconName}.");
                }
                else
                {
                    Console.WriteLine($"Could not create Lexicon: {lexiconName}.");
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine("Exception caught: " + ex.Message);
            }
        }
    }
```

- API の詳細については、「 API リファレンスPutLexicon」の「」を参照してください。 AWS SDK for .NET

`SynthesizeSpeech`

次の例は、SynthesizeSpeech を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    using System;
    using System.IO;
    using System.Threading.Tasks;
    using Amazon.Polly;
    using Amazon.Polly.Model;

    public class SynthesizeSpeech
    {
        public static async Task Main()
```

```
        {
            string outputFileName = "speech.mp3";
            string text = "Twas brillig, and the slithy toves did gyre and gimbol in
the wabe";

            var client = new AmazonPollyClient();
            var response = await PollySynthesizeSpeech(client, text);

            WriteSpeechToStream(response.AudioStream, outputFileName);
        }

        /// <summary>
        /// Calls the Amazon Polly SynthesizeSpeechAsync method to convert text
        /// to speech.
        /// </summary>
        /// <param name="client">The Amazon Polly client object used to connect
        /// to the Amazon Polly service.</param>
        /// <param name="text">The text to convert to speech.</param>
        /// <returns>A SynthesizeSpeechResponse object that includes an AudioStream
        /// object with the converted text.</returns>
        private static async Task<SynthesizeSpeechResponse>
PollySynthesizeSpeech(IAmazonPolly client, string text)
        {
            var synthesizeSpeechRequest = new SynthesizeSpeechRequest()
            {
                OutputFormat = OutputFormat.Mp3,
                VoiceId = VoiceId.Joanna,
                Text = text,
            };

            var synthesizeSpeechResponse =
                await client.SynthesizeSpeechAsync(synthesizeSpeechRequest);

            return synthesizeSpeechResponse;
        }

        /// <summary>
        /// Writes the AudioStream returned from the call to
        /// SynthesizeSpeechAsync to a file in MP3 format.
        /// </summary>
        /// <param name="audioStream">The AudioStream returned from the
        /// call to the SynthesizeSpeechAsync method.</param>
        /// <param name="outputFileName">The full path to the file in which to
        /// save the audio stream.</param>
```

```
        private static void WriteSpeechToStream(Stream audioStream, string
outputFileName)
        {
            var outputStream = new FileStream(
                outputFileName,
                FileMode.Create,
                FileAccess.Write);
            byte[] buffer = new byte[2 * 1024];
            int readBytes;

            while ((readBytes = audioStream.Read(buffer, 0, 2 * 1024)) > 0)
            {
                outputStream.Write(buffer, 0, readBytes);
            }

            // Flushes the buffer to avoid losing the last second or so of
            // the synthesized text.
            outputStream.Flush();
            Console.WriteLine($"Saved {outputFileName} to disk.");
        }
    }
```

AWS SDK を使用して Amazon Polly で音声マークを使用してテキストから音声を合成します。

```
    using System;
    using System.Collections.Generic;
    using System.IO;
    using System.Threading.Tasks;
    using Amazon.Polly;
    using Amazon.Polly.Model;

    public class SynthesizeSpeechMarks
    {
        public static async Task Main()
        {
            var client = new AmazonPollyClient();
            string outputFileName = "speechMarks.json";

            var synthesizeSpeechRequest = new SynthesizeSpeechRequest()
            {
                OutputFormat = OutputFormat.Json,
```

```
                SpeechMarkTypes = new List<string>
                {
                    SpeechMarkType.Viseme,
                    SpeechMarkType.Word,
                },
                VoiceId = VoiceId.Joanna,
                Text = "This is a sample text to be synthesized.",
            };

            try
            {
                using (var outputStream = new FileStream(outputFileName,
FileMode.Create, FileAccess.Write))
                {
                    var synthesizeSpeechResponse = await
client.SynthesizeSpeechAsync(synthesizeSpeechRequest);
                    var buffer = new byte[2 * 1024];
                    int readBytes;

                    var inputStream = synthesizeSpeechResponse.AudioStream;
                    while ((readBytes = inputStream.Read(buffer, 0, 2 * 1024)) > 0)
                    {
                        outputStream.Write(buffer, 0, readBytes);
                    }
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Error: {ex.Message}");
            }
        }
    }
```

- API の詳細については、「 API リファレンスSynthesizeSpeech」の「」を参照してください。AWS SDK for .NET

を使用した Amazon RDS の例 AWS SDK for .NET

次のコード例は、Amazon RDS AWS SDK for .NET で 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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。このリンクには GitHub、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が記載されています。

開始方法

Hello Amazon RDS

次のコード例は、Amazon RDS の使用を開始する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

namespace RDSActions;

public static class HelloRds
{
    static async Task Main(string[] args)
    {
        var rdsClient = new AmazonRDSClient();

        Console.WriteLine($"Hello Amazon RDS! Following are some of your DB
 instances:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
```

```
        // Let's get the first twenty DB instances.
        var response = await rdsClient.DescribeDBInstancesAsync(
            new DescribeDBInstancesRequest()
            {
                MaxRecords = 20 // Must be between 20 and 100.
            });

        foreach (var instance in response.DBInstances)
        {
            Console.WriteLine($"\tDB name: {instance.DBName}");
            Console.WriteLine($"\tArn: {instance.DBInstanceArn}");
            Console.WriteLine($"\tIdentifier: {instance.DBInstanceIdentifier}");
            Console.WriteLine();
        }
    }
}
```

- 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。

トピック

- アクション
- シナリオ

アクション

`CreateDBInstance`

次の例は、`CreateDBInstance` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
   /// <summary>
   /// Create an 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="dbName">Name for the DB instance.</param>
   /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
   /// <param name="parameterGroupName">DB parameter group to associate with the
instance.</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>
   /// <param name="allocatedStorage">The amount of storage in gibibytes (GiB) to
allocate to the DB instance.</param>
   /// <param name="adminName">Admin user name.</param>
   /// <param name="adminPassword">Admin user password.</param>
   /// <returns>DB instance object.</returns>
   public async Task<DBInstance> CreateDBInstance(string dbName, string
dbInstanceIdentifier,
       string parameterGroupName, string dbEngine, string dbEngineVersion,
       string instanceClass, int allocatedStorage, string adminName, string
adminPassword)
   {
       var response = await _amazonRDS.CreateDBInstanceAsync(
           new CreateDBInstanceRequest()
           {
               DBName = dbName,
               DBInstanceIdentifier = dbInstanceIdentifier,
               DBParameterGroupName = parameterGroupName,
               Engine = dbEngine,
               EngineVersion = dbEngineVersion,
               DBInstanceClass = instanceClass,
               AllocatedStorage = allocatedStorage,
               MasterUsername = adminName,
               MasterUserPassword = adminPassword
           });

       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。

`CreateDBParameterGroup`

次の例は、`CreateDBParameterGroup` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
   /// <summary>
   /// Create a new DB parameter group. Use the action
DescribeDBParameterGroupsAsync
   /// to determine when the DB parameter group is ready to use.
   /// </summary>
   /// <param name="name">Name of the DB parameter group.</param>
   /// <param name="family">Family of the DB parameter group.</param>
   /// <param name="description">Description of the DB parameter group.</param>
   /// <returns>The new DB parameter group.</returns>
   public async Task<DBParameterGroup> CreateDBParameterGroup(
       string name, string family, string description)
   {
       var response = await _amazonRDS.CreateDBParameterGroupAsync(
           new CreateDBParameterGroupRequest()
           {
               DBParameterGroupName = name,
               DBParameterGroupFamily = family,
               Description = description
           });
       return response.DBParameterGroup;
   }
```

- API の詳細については、「 API リファレンスCreateDBParameterGroup」を参照してください。 AWS SDK for .NET

`CreateDBSnapshot`

次の例は、`CreateDBSnapshot` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Create a snapshot of a DB instance.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
    /// <returns>DB snapshot object.</returns>
    public async Task<DBSnapshot> CreateDBSnapshot(string dbInstanceIdentifier,
string snapshotIdentifier)
    {
        var response = await _amazonRDS.CreateDBSnapshotAsync(
            new CreateDBSnapshotRequest()
            {
                DBSnapshotIdentifier = snapshotIdentifier,
                DBInstanceIdentifier = dbInstanceIdentifier
            });

        return response.DBSnapshot;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「CreateDBSnapshot」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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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> DeleteDBInstance(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。

DeleteDBParameterGroup

次の例は、DeleteDBParameterGroup を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete a DB parameter group. The group cannot be a default DB parameter
group
    /// or be associated with any DB instances.
    /// </summary>
    /// <param name="name">Name of the DB parameter group.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteDBParameterGroup(string name)
    {
        var response = await _amazonRDS.DeleteDBParameterGroupAsync(
            new DeleteDBParameterGroupRequest()
            {
                DBParameterGroupName = name,
            });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスDeleteDBParameterGroup」を参照してください。 AWS SDK for .NET

`DescribeDBEngineVersions`

次の例は、`DescribeDBEngineVersions` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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">Name of the engine.</param>
    /// <param name="dbParameterGroupFamily">Optional parameter group family name.</
param>
    /// <returns>List of DBEngineVersions.</returns>
    public async Task<List<DBEngineVersion>> DescribeDBEngineVersions(string engine,
        string dbParameterGroupFamily = null)
    {
        var response = await _amazonRDS.DescribeDBEngineVersionsAsync(
            new DescribeDBEngineVersionsRequest()
            {
                Engine = engine,
                DBParameterGroupFamily = dbParameterGroupFamily
            });
        return response.DBEngineVersions;
    }
```

- API の詳細については、「 API DescribeDBEngineVersionsAWS SDK for .NET 」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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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>> DescribeDBInstances(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。

`DescribeDBParameterGroups`

次の例は、`DescribeDBParameterGroups` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get descriptions of DB parameter groups.
    /// </summary>
    /// <param name="name">Optional name of the DB parameter group to describe.</
param>
    /// <returns>The list of DB parameter group descriptions.</returns>
    public async Task<List<DBParameterGroup>> DescribeDBParameterGroups(string name
 = null)
    {
        var response = await _amazonRDS.DescribeDBParameterGroupsAsync(
            new DescribeDBParameterGroupsRequest()
            {
                DBParameterGroupName = name
            });
        return response.DBParameterGroups;
    }
```

- API の詳細については、「 API DescribeDBParameterGroupsAWS SDK for .NET 」を参照してください。

`DescribeDBParameters`

次の例は、`DescribeDBParameters` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get a list of DB parameters from a specific parameter group.
    /// </summary>
    /// <param name="dbParameterGroupName">Name of a specific DB parameter group.</
param>
    /// <param name="source">Optional source for selecting parameters.</param>
    /// <returns>List of parameter values.</returns>
    public async Task<List<Parameter>> DescribeDBParameters(string
 dbParameterGroupName, string source = null)
    {
        var results = new List<Parameter>();
        var paginateParameters = _amazonRDS.Paginators.DescribeDBParameters(
            new DescribeDBParametersRequest()
            {
                DBParameterGroupName = dbParameterGroupName,
                Source = source
            });
        // Get the entire list using the paginator.
        await foreach (var parameters in paginateParameters.Parameters)
        {
            results.Add(parameters);
        }
        return results;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DescribeDBParameters」を参照してください。

DescribeDBSnapshots

次の例は、DescribeDBSnapshots を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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 instance.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <returns>List of DB snapshots.</returns>
    public async Task<List<DBSnapshot>> DescribeDBSnapshots(string
dbInstanceIdentifier)
    {
        var results = new List<DBSnapshot>();
        var snapshotsPaginator = _amazonRDS.Paginators.DescribeDBSnapshots(
            new DescribeDBSnapshotsRequest()
            {
                DBInstanceIdentifier = dbInstanceIdentifier
            });

        // Get the entire list using the paginator.
        await foreach (var snapshots in snapshotsPaginator.DBSnapshots)
        {
            results.Add(snapshots);
        }
        return results;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DescribeDBSnapshots」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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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>>
DescribeOrderableDBInstanceOptions(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、「 API リファレンス」のDescribeOrderable「DBInstanceOptions」を参照してください。 AWS SDK for .NET

`ModifyDBParameterGroup`

次の例は、ModifyDBParameterGroup を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Update a DB parameter group. Use the action DescribeDBParameterGroupsAsync
    /// to determine when the DB parameter group is ready to use.
    /// </summary>
    /// <param name="name">Name of the DB parameter group.</param>
    /// <param name="parameters">List of parameters. Maximum of 20 per request.</
param>
    /// <returns>The updated DB parameter group name.</returns>
    public async Task<string> ModifyDBParameterGroup(
        string name, List<Parameter> parameters)
    {
        var response = await _amazonRDS.ModifyDBParameterGroupAsync(
            new ModifyDBParameterGroupRequest()
            {
                DBParameterGroupName = name,
                Parameters = parameters,
            });
        return response.DBParameterGroupName;
    }
```

- API の詳細については、「 API リファレンスModifyDBParameterGroup」を参照してください。 AWS SDK for .NET

シナリオ

DB インスタンスの使用を開始する

次のコードサンプルは、以下の操作方法を示しています。

- カスタム DB 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ように設定した DB インスタンスを作成します。DB インスタンスにはデータベースも含まれています。
- インスタンスのスナップショットを取得します。
- インスタンスとパラメータグループを削除します。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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のシナリオを実行します。

```
/// <summary>
/// Scenario for RDS DB instance example.
/// </summary>
public class RDSInstanceScenario
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.

    This .NET example performs the following tasks:
    1.  Returns a list of the available DB engine families using the
 DescribeDBEngineVersionsAsync method.
    2.  Selects an engine family and creates a custom DB parameter group using the
 CreateDBParameterGroupAsync method.
    3.  Gets the parameter groups using the DescribeDBParameterGroupsAsync method.
    4.  Gets parameters in the group using the DescribeDBParameters method.
    5.  Parses and displays parameters in the group.
```

```
   6.  Modifies both the auto_increment_offset and auto_increment_increment
parameters
       using the ModifyDBParameterGroupAsync method.
   7.  Gets and displays the updated parameters using the DescribeDBParameters
method with a source of "user".
   8.  Gets a list of allowed engine versions using the
DescribeDBEngineVersionsAsync method.
   9.  Displays and selects from a list of micro instance classes available for the
selected engine and version.
   10. Creates an RDS DB instance that contains a MySql database and uses the
parameter group
       using the CreateDBInstanceAsync method.
   11. Waits for DB instance to be ready using the DescribeDBInstancesAsync method.
   12. Prints out the connection endpoint string for the new DB instance.
   13. Creates a snapshot of the DB instance using the CreateDBSnapshotAsync
method.
   14. Waits for DB snapshot to be ready using the DescribeDBSnapshots method.
   15. Deletes the DB instance using the DeleteDBInstanceAsync method.
   16. Waits for DB instance to be deleted using the DescribeDbInstances method.
   17. Deletes the parameter group using the DeleteDBParameterGroupAsync.
   */

   private static readonly string sepBar = new('-', 80);
   private static RDSWrapper rdsWrapper = null!;
   private static ILogger logger = null!;
   private static readonly string engine = "mysql";
   static async Task Main(string[] args)
   {
       // Set up dependency injection for the Amazon RDS 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<IAmazonRDS>()
                   .AddTransient<RDSWrapper>()
           )
           .Build();

       logger = LoggerFactory.Create(builder =>
       {
           builder.AddConsole();
```

```
        }).CreateLogger<RDSInstanceScenario>();

        rdsWrapper = host.Services.GetRequiredService<RDSWrapper>();

        Console.WriteLine(sepBar);
        Console.WriteLine(
            "Welcome to the Amazon Relational Database Service (Amazon RDS) DB
instance scenario example.");
        Console.WriteLine(sepBar);

        try
        {
            var parameterGroupFamily = await ChooseParameterGroupFamily();

            var parameterGroup = await CreateDbParameterGroup(parameterGroupFamily);

            var parameters = await
DescribeParametersInGroup(parameterGroup.DBParameterGroupName,
                new List<string> { "auto_increment_offset",
"auto_increment_increment" });

            await ModifyParameters(parameterGroup.DBParameterGroupName, parameters);

            await DescribeUserSourceParameters(parameterGroup.DBParameterGroupName);

            var engineVersionChoice = await
ChooseDbEngineVersion(parameterGroupFamily);

            var instanceChoice = await ChooseDbInstanceClass(engine,
engineVersionChoice.EngineVersion);

            var newInstanceIdentifier = "Example-Instance-" + DateTime.Now.Ticks;

            var newInstance = await CreateRdsNewInstance(parameterGroup, engine,
engineVersionChoice.EngineVersion,
                instanceChoice.DBInstanceClass, newInstanceIdentifier);
            if (newInstance != null)
            {
                DisplayConnectionString(newInstance);

                await CreateSnapshot(newInstance);

                await DeleteRdsInstance(newInstance);
            }
```

```
            await DeleteParameterGroup(parameterGroup);

            Console.WriteLine("Scenario complete.");
            Console.WriteLine(sepBar);
        }
        catch (Exception ex)
        {
            logger.LogError(ex, "There was a problem executing the scenario.");
        }
    }

    /// <summary>
    /// Choose the RDS DB parameter group family from a list of available options.
    /// </summary>
    /// <returns>The selected parameter group family.</returns>
    public static async Task<string> ChooseParameterGroupFamily()
    {
        Console.WriteLine(sepBar);
        // 1. Get a list of available engines.
        var engines = await rdsWrapper.DescribeDBEngineVersions(engine);

        Console.WriteLine("1. The following is a list of available DB parameter
group families:");
        int i = 1;
        var parameterGroupFamilies = engines.GroupBy(e =>
e.DBParameterGroupFamily).ToList();
        foreach (var parameterGroupFamily in parameterGroupFamilies)
        {
            // List the available parameter group families.
            Console.WriteLine(
                $"\t{i}. Family: {parameterGroupFamily.Key}");
            i++;
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > parameterGroupFamilies.Count)
        {
            Console.WriteLine("Select an available DB parameter group family by
entering a number from the list above:");
            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<DBParameterGroup> CreateDbParameterGroup(string
dbParameterGroupFamily)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"2. Create new DB parameter group with family
{dbParameterGroupFamily}:");

        var parameterGroup = await rdsWrapper.CreateDBParameterGroup(
            "ExampleParameterGroup-" + DateTime.Now.Ticks,
            dbParameterGroupFamily, "New example parameter group");

        var groupInfo =
            await rdsWrapper.DescribeDBParameterGroups(parameterGroup
                .DBParameterGroupName);

        Console.WriteLine(
            $"3. New DB parameter group: \n\t{groupInfo[0].Description}, \n\tARN
{groupInfo[0].DBParameterGroupArn}");
        Console.WriteLine(sepBar);
        return parameterGroup;
    }

    /// <summary>
    /// Get and describe parameters from a DBParameterGroup.
    /// </summary>
    /// <param name="parameterGroupName">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>> DescribeParametersInGroup(string
parameterGroupName, List<string>? parameterNames = null)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("4. Get some parameters from the group.");
```

```
        Console.WriteLine(sepBar);

        var parameters =
            await rdsWrapper.DescribeDBParameters(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 ModifyParameters(string parameterGroupName,
List<Parameter> parameters)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("6. Modify some parameters in the group.");

        foreach (var p in parameters)
        {
            if (p.IsModifiable && p.DataType == "integer")
            {
                int newValue = 0;
                while (newValue == 0)
                {
                    Console.WriteLine(
                        $"Enter a new value for {p.ParameterName} from the allowed
values {p.AllowedValues} ");
```

```
                    var choice = Console.ReadLine();
                    Int32.TryParse(choice, out newValue);
                }

                p.ParameterValue = newValue.ToString();
            }
        }

        await rdsWrapper.ModifyDBParameterGroup(parameterGroupName, parameters);

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Describe the user source parameters in the group.
    /// </summary>
    /// <param name="parameterGroupName">Name of the DBParameterGroup.</param>
    /// <returns>Async task.</returns>
    public static async Task DescribeUserSourceParameters(string parameterGroupName)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("7. Describe user source parameters in the group.");

        var parameters =
            await rdsWrapper.DescribeDBParameters(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> ChooseDbEngineVersion(string
 dbParameterGroupFamily)
    {
        Console.WriteLine(sepBar);
        // Get a list of allowed engines.
        var allowedEngines =
            await rdsWrapper.DescribeDBEngineVersions(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}. Engine: {version.Engine} Version
 {version.EngineVersion}.");
            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>
    /// 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 rdsWrapper.DescribeOrderableDBInstanceOptions(engine,
 engineVersion);

        Console.WriteLine($"8. Available micro DB instance classes for engine
 {engine} and version {engineVersion}:");
        int i = 1;

        // Filter to micro instances for this example.
        allowedInstances = allowedInstances
            .Where(i => i.DBInstanceClass.Contains("micro")).ToList();

        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("9. Select an available DB instance class by entering
 a number from the list above:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var instanceChoice = allowedInstances[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return instanceChoice;
    }

    /// <summary>
    /// Create a new RDS DB instance.
    /// </summary>
    /// <param name="parameterGroup">Parameter group to use for the DB instance.</
param>
    /// <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?> CreateRdsNewInstance(DBParameterGroup
 parameterGroup,
        string engineName, string engineVersion, string instanceClass, string
 instanceIdentifier)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"10. Create a new DB instance with identifier
 {instanceIdentifier}.");
        bool isInstanceReady = false;
        DBInstance newInstance;
        var instances = await rdsWrapper.DescribeDBInstances();
        isInstanceReady = instances.FirstOrDefault(i =>
            i.DBInstanceIdentifier == instanceIdentifier)?.DBInstanceStatus ==
 "available";

        if (isInstanceReady)
        {
            Console.WriteLine("Instance already created.");
            newInstance = instances.First(i => i.DBInstanceIdentifier ==
 instanceIdentifier);
        }
        else
        {
            Console.WriteLine("Please enter an admin user name:");
            var username = Console.ReadLine();

            Console.WriteLine("Please enter an admin password:");
            var password = Console.ReadLine();

            newInstance = await rdsWrapper.CreateDBInstance(
                "ExampleInstance",
                instanceIdentifier,
                parameterGroup.DBParameterGroupName,
                engineName,
                engineVersion,
                instanceClass,
                20,
```

```
                username,
                password
            );

            // 11. Wait for the DB instance to be ready.

            Console.WriteLine("11. Waiting for DB instance to be ready...");
            while (!isInstanceReady)
            {
                instances = await
 rdsWrapper.DescribeDBInstances(instanceIdentifier);
                isInstanceReady = instances.FirstOrDefault()?.DBInstanceStatus ==
 "available";
                newInstance = instances.First();
                Thread.Sleep(30000);
            }
        }

        Console.WriteLine(sepBar);
        return newInstance;
    }

    /// <summary>
    /// Display a connection string for an RDS DB instance.
    /// </summary>
    /// <param name="instance">The DB instance to use to get a connection string.</
param>
    public static void DisplayConnectionString(DBInstance instance)
    {
        Console.WriteLine(sepBar);
        // Display the connection string.
        Console.WriteLine("12. New DB instance connection string: ");
        Console.WriteLine(
            $"\n{engine} -h {instance.Endpoint.Address} -P {instance.Endpoint.Port}
 "
            + $"-u {instance.MasterUsername} -p [YOUR PASSWORD]\n");

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Create a snapshot from an RDS DB instance.
    /// </summary>
    /// <param name="instance">DB instance to use when creating a snapshot.</param>
```

```
    /// <returns>The snapshot object.</returns>
    public static async Task<DBSnapshot> CreateSnapshot(DBInstance instance)
    {
        Console.WriteLine(sepBar);
        // Create a snapshot.
        Console.WriteLine($"13. Creating snapshot from DB instance
{instance.DBInstanceIdentifier}.");
        var snapshot = await
rdsWrapper.CreateDBSnapshot(instance.DBInstanceIdentifier, "ExampleSnapshot-" +
DateTime.Now.Ticks);

        // Wait for the snapshot to be available
        bool isSnapshotReady = false;

        Console.WriteLine($"14. Waiting for snapshot to be ready...");
        while (!isSnapshotReady)
        {
            var snapshots = await
rdsWrapper.DescribeDBSnapshots(instance.DBInstanceIdentifier);
            isSnapshotReady = snapshots.FirstOrDefault()?.Status == "available";
            snapshot = snapshots.First();
            Thread.Sleep(30000);
        }

        Console.WriteLine(
            $"Snapshot {snapshot.DBSnapshotIdentifier} status is
{snapshot.Status}.");
        Console.WriteLine(sepBar);
        return snapshot;
    }

    /// <summary>
    /// Delete an RDS DB instance.
    /// </summary>
    /// <param name="instance">The DB instance to delete.</param>
    /// <returns>Async task.</returns>
    public static async Task DeleteRdsInstance(DBInstance newInstance)
    {
        Console.WriteLine(sepBar);
        // Delete the DB instance.
        Console.WriteLine($"15. Delete the DB instance
{newInstance.DBInstanceIdentifier}.");
        await rdsWrapper.DeleteDBInstance(newInstance.DBInstanceIdentifier);
```

```
        // Wait for the DB instance to delete.
        Console.WriteLine($"16. Waiting for the DB instance to delete...");
        bool isInstanceDeleted = false;

        while (!isInstanceDeleted)
        {
            var instance = await rdsWrapper.DescribeDBInstances();
            isInstanceDeleted = instance.All(i => i.DBInstanceIdentifier !=
 newInstance.DBInstanceIdentifier);
            Thread.Sleep(30000);
        }

        Console.WriteLine("DB instance deleted.");
        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Delete a DB parameter group.
    /// </summary>
    /// <param name="parameterGroup">The parameter group to delete.</param>
    /// <returns>Async task.</returns>
    public static async Task DeleteParameterGroup(DBParameterGroup parameterGroup)
    {
        Console.WriteLine(sepBar);
        // Delete the parameter group.
        Console.WriteLine($"17. Delete the DB parameter group
 {parameterGroup.DBParameterGroupName}.");
        await
 rdsWrapper.DeleteDBParameterGroup(parameterGroup.DBParameterGroupName);

        Console.WriteLine(sepBar);
    }
```

DB インスタンスアクションのシナリオで使用されるラッパーメソッド。

```
/// <summary>
/// Wrapper methods to use Amazon Relational Database Service (Amazon RDS) with DB
 instance operations.
/// </summary>
public partial class RDSWrapper
{
```

```
    private readonly IAmazonRDS _amazonRDS;
    public RDSWrapper(IAmazonRDS amazonRDS)
    {
        _amazonRDS = amazonRDS;
    }


    /// <summary>
    /// Get a list of DB engine versions for a particular DB engine.
    /// </summary>
    /// <param name="engine">Name of the engine.</param>
    /// <param name="dbParameterGroupFamily">Optional parameter group family name.</
param>
    /// <returns>List of DBEngineVersions.</returns>
    public async Task<List<DBEngineVersion>> DescribeDBEngineVersions(string engine,
        string dbParameterGroupFamily = null)
    {
        var response = await _amazonRDS.DescribeDBEngineVersionsAsync(
            new DescribeDBEngineVersionsRequest()
            {
                Engine = engine,
                DBParameterGroupFamily = dbParameterGroupFamily
            });
        return response.DBEngineVersions;
    }



    /// <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>>
 DescribeOrderableDBInstanceOptions(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>
    /// 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>> DescribeDBInstances(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>
    /// Create an 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="dbName">Name for the DB instance.</param>
   /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
   /// <param name="parameterGroupName">DB parameter group to associate with the
instance.</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>
   /// <param name="allocatedStorage">The amount of storage in gibibytes (GiB) to
allocate to the DB instance.</param>
   /// <param name="adminName">Admin user name.</param>
   /// <param name="adminPassword">Admin user password.</param>
   /// <returns>DB instance object.</returns>
   public async Task<DBInstance> CreateDBInstance(string dbName, string
dbInstanceIdentifier,
       string parameterGroupName, string dbEngine, string dbEngineVersion,
       string instanceClass, int allocatedStorage, string adminName, string
adminPassword)
   {
       var response = await _amazonRDS.CreateDBInstanceAsync(
           new CreateDBInstanceRequest()
           {
               DBName = dbName,
               DBInstanceIdentifier = dbInstanceIdentifier,
               DBParameterGroupName = parameterGroupName,
               Engine = dbEngine,
               EngineVersion = dbEngineVersion,
               DBInstanceClass = instanceClass,
               AllocatedStorage = allocatedStorage,
               MasterUsername = adminName,
               MasterUserPassword = adminPassword
           });

       return response.DBInstance;
   }



   /// <summary>
   /// Delete a particular DB instance.
   /// </summary>
   /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
   /// <returns>DB instance object.</returns>
   public async Task<DBInstance> DeleteDBInstance(string dbInstanceIdentifier)
   {
```

```
        var response = await _amazonRDS.DeleteDBInstanceAsync(
            new DeleteDBInstanceRequest()
            {
                DBInstanceIdentifier = dbInstanceIdentifier,
                SkipFinalSnapshot = true,
                DeleteAutomatedBackups = true
            });

        return response.DBInstance;
    }
```

DB パラメータグループのシナリオで使用されるラッパーメソッド。

```
/// <summary>
/// Wrapper methods to use Amazon Relational Database Service (Amazon RDS) with
 parameter groups.
/// </summary>
public partial class RDSWrapper
{

    /// <summary>
    /// Get descriptions of DB parameter groups.
    /// </summary>
    /// <param name="name">Optional name of the DB parameter group to describe.</
param>
    /// <returns>The list of DB parameter group descriptions.</returns>
    public async Task<List<DBParameterGroup>> DescribeDBParameterGroups(string name
 = null)
    {
        var response = await _amazonRDS.DescribeDBParameterGroupsAsync(
            new DescribeDBParameterGroupsRequest()
            {
                DBParameterGroupName = name
            });
        return response.DBParameterGroups;
    }



    /// <summary>
```

```
    /// Create a new DB parameter group. Use the action
 DescribeDBParameterGroupsAsync
    /// to determine when the DB parameter group is ready to use.
    /// </summary>
    /// <param name="name">Name of the DB parameter group.</param>
    /// <param name="family">Family of the DB parameter group.</param>
    /// <param name="description">Description of the DB parameter group.</param>
    /// <returns>The new DB parameter group.</returns>
    public async Task<DBParameterGroup> CreateDBParameterGroup(
        string name, string family, string description)
    {
        var response = await _amazonRDS.CreateDBParameterGroupAsync(
            new CreateDBParameterGroupRequest()
            {
                DBParameterGroupName = name,
                DBParameterGroupFamily = family,
                Description = description
            });
        return response.DBParameterGroup;
    }



    /// <summary>
    /// Update a DB parameter group. Use the action DescribeDBParameterGroupsAsync
    /// to determine when the DB parameter group is ready to use.
    /// </summary>
    /// <param name="name">Name of the DB parameter group.</param>
    /// <param name="parameters">List of parameters. Maximum of 20 per request.</
param>
    /// <returns>The updated DB parameter group name.</returns>
    public async Task<string> ModifyDBParameterGroup(
        string name, List<Parameter> parameters)
    {
        var response = await _amazonRDS.ModifyDBParameterGroupAsync(
            new ModifyDBParameterGroupRequest()
            {
                DBParameterGroupName = name,
                Parameters = parameters,
            });
        return response.DBParameterGroupName;
    }
```

```
    /// <summary>
    /// Delete a DB parameter group. The group cannot be a default DB parameter
 group
    /// or be associated with any DB instances.
    /// </summary>
    /// <param name="name">Name of the DB parameter group.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteDBParameterGroup(string name)
    {
        var response = await _amazonRDS.DeleteDBParameterGroupAsync(
            new DeleteDBParameterGroupRequest()
            {
                DBParameterGroupName = name,
            });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }



    /// <summary>
    /// Get a list of DB parameters from a specific parameter group.
    /// </summary>
    /// <param name="dbParameterGroupName">Name of a specific DB parameter group.</
param>
    /// <param name="source">Optional source for selecting parameters.</param>
    /// <returns>List of parameter values.</returns>
    public async Task<List<Parameter>> DescribeDBParameters(string
 dbParameterGroupName, string source = null)
    {
        var results = new List<Parameter>();
        var paginateParameters = _amazonRDS.Paginators.DescribeDBParameters(
            new DescribeDBParametersRequest()
            {
                DBParameterGroupName = dbParameterGroupName,
                Source = source
            });
        // Get the entire list using the paginator.
        await foreach (var parameters in paginateParameters.Parameters)
        {
            results.Add(parameters);
        }
        return results;
    }
```

DB スナップショットアクションのシナリオで使用されるラッパーメソッド。

```
/// <summary>
/// Wrapper methods to use Amazon Relational Database Service (Amazon RDS) with
 snapshots.
/// </summary>
public partial class RDSWrapper
{

    /// <summary>
    /// Create a snapshot of a DB instance.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
    /// <returns>DB snapshot object.</returns>
    public async Task<DBSnapshot> CreateDBSnapshot(string dbInstanceIdentifier,
 string snapshotIdentifier)
    {
        var response = await _amazonRDS.CreateDBSnapshotAsync(
            new CreateDBSnapshotRequest()
            {
                DBSnapshotIdentifier = snapshotIdentifier,
                DBInstanceIdentifier = dbInstanceIdentifier
            });

        return response.DBSnapshot;
    }



    /// <summary>
    /// Return a list of DB snapshots for a particular DB instance.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <returns>List of DB snapshots.</returns>
    public async Task<List<DBSnapshot>> DescribeDBSnapshots(string
 dbInstanceIdentifier)
    {
        var results = new List<DBSnapshot>();
```

```
        var snapshotsPaginator = _amazonRDS.Paginators.DescribeDBSnapshots(
            new DescribeDBSnapshotsRequest()
            {
                DBInstanceIdentifier = dbInstanceIdentifier
            });

        // Get the entire list using the paginator.
        await foreach (var snapshots in snapshotsPaginator.DBSnapshots)
        {
            results.Add(snapshots);
        }
        return results;
    }
```

- 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。
  - CreateDBInstance
  - CreateDBParameterGroup
  - CreateDBSnapshot
  - DeleteDBInstance
  - DeleteDBParameterGroup
  - DescribeDBEngineVersions
  - DescribeDBInstances
  - DescribeDBParameterGroups
  - DescribeDBParameters
  - DescribeDBSnapshots
  - DescribeOrderableDBInstanceOptions
  - ModifyDBParameterGroup

を使用した Amazon Rekognition の例 AWS SDK for .NET

次のコード例は、Amazon Rekognition AWS SDK for .NET で 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。 Amazon Rekognition

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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。このリンクには GitHub、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が記載されています。

トピック

- アクション

アクション

`CompareFaces`

次の例は、CompareFaces を使用する方法を説明しています。

詳細については、「イメージ内の顔を比較する」を参照してください。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
using System;
using System.IO;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to compare faces in two images.
/// </summary>
public class CompareFaces
{
    public static async Task Main()
```

```
        {
            float similarityThreshold = 70F;
            string sourceImage = "source.jpg";
            string targetImage = "target.jpg";

            var rekognitionClient = new AmazonRekognitionClient();

            Amazon.Rekognition.Model.Image imageSource = new
Amazon.Rekognition.Model.Image();

            try
            {
                using FileStream fs = new FileStream(sourceImage, FileMode.Open,
FileAccess.Read);
                byte[] data = new byte[fs.Length];
                fs.Read(data, 0, (int)fs.Length);
                imageSource.Bytes = new MemoryStream(data);
            }
            catch (Exception)
            {
                Console.WriteLine($"Failed to load source image: {sourceImage}");
                return;
            }

            Amazon.Rekognition.Model.Image imageTarget = new
Amazon.Rekognition.Model.Image();

            try
            {
                using FileStream fs = new FileStream(targetImage, FileMode.Open,
FileAccess.Read);
                byte[] data = new byte[fs.Length];
                data = new byte[fs.Length];
                fs.Read(data, 0, (int)fs.Length);
                imageTarget.Bytes = new MemoryStream(data);
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Failed to load target image: {targetImage}");
                Console.WriteLine(ex.Message);
                return;
            }

            var compareFacesRequest = new CompareFacesRequest
```

```
            {
                SourceImage = imageSource,
                TargetImage = imageTarget,
                SimilarityThreshold = similarityThreshold,
            };

            // Call operation
            var compareFacesResponse = await
rekognitionClient.CompareFacesAsync(compareFacesRequest);

            // Display results
            compareFacesResponse.FaceMatches.ForEach(match =>
            {
                ComparedFace face = match.Face;
                BoundingBox position = face.BoundingBox;
                Console.WriteLine($"Face at {position.Left} {position.Top} matches
with {match.Similarity}% confidence.");
            });

            Console.WriteLine($"Found {compareFacesResponse.UnmatchedFaces.Count}
face(s) that did not match.");
        }
    }
```

- API の詳細については、「 API リファレンスCompareFaces」の「」を参照してください。 AWS SDK for .NET

`CreateCollection`

次の例は、`CreateCollection` を使用する方法を説明しています。

詳細については、「コレクションを作成する」を参照してください。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

    /// <summary>
    /// Uses Amazon Rekognition to create a collection to which you can add
    /// faces using the IndexFaces operation.
    /// </summary>
    public class CreateCollection
    {
        public static async Task Main()
        {
            var rekognitionClient = new AmazonRekognitionClient();

            string collectionId = "MyCollection";
            Console.WriteLine("Creating collection: " + collectionId);

            var createCollectionRequest = new CreateCollectionRequest
            {
                CollectionId = collectionId,
            };

            CreateCollectionResponse createCollectionResponse = await
rekognitionClient.CreateCollectionAsync(createCollectionRequest);
            Console.WriteLine($"CollectionArn :
{createCollectionResponse.CollectionArn}");
            Console.WriteLine($"Status code :
{createCollectionResponse.StatusCode}");
        }
    }
```

- API の詳細については、「 API リファレンスCreateCollection」の「」を参照してください。 AWS SDK for .NET

DeleteCollection

次の例は、DeleteCollection を使用する方法を説明しています。

詳細については、「コレクションを削除する」を参照してください。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

    /// <summary>
    /// Uses the Amazon Rekognition Service to delete an existing collection.
    /// </summary>
    public class DeleteCollection
    {
        public static async Task Main()
        {
            var rekognitionClient = new AmazonRekognitionClient();

            string collectionId = "MyCollection";
            Console.WriteLine("Deleting collection: " + collectionId);

            var deleteCollectionRequest = new DeleteCollectionRequest()
            {
                CollectionId = collectionId,
            };

            var deleteCollectionResponse = await
rekognitionClient.DeleteCollectionAsync(deleteCollectionRequest);
            Console.WriteLine($"{collectionId}:
{deleteCollectionResponse.StatusCode}");
        }
    }
```

- API の詳細については、「 API リファレンスDeleteCollection」の「」を参照してください。AWS SDK for .NET

DeleteFaces

次の例は、DeleteFaces を使用する方法を説明しています。

詳細については、「コレクションから顔を削除する」を参照してください。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses the Amazon Rekognition Service to delete one or more faces from
    /// a Rekognition collection.
    /// </summary>
    public class DeleteFaces
    {
        public static async Task Main()
        {
            string collectionId = "MyCollection";
            var faces = new List<string> { "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx" };

            var rekognitionClient = new AmazonRekognitionClient();

            var deleteFacesRequest = new DeleteFacesRequest()
            {
                CollectionId = collectionId,
                FaceIds = faces,
            };

            DeleteFacesResponse deleteFacesResponse = await
rekognitionClient.DeleteFacesAsync(deleteFacesRequest);
            deleteFacesResponse.DeletedFaces.ForEach(face =>
            {
```

```
                Console.WriteLine($"FaceID: {face}");
            });
        }
    }
```

- API の詳細については、「 API リファレンスDeleteFaces」の「」を参照してください。 AWS SDK for .NET

`DescribeCollection`

次の例は、DescribeCollection を使用する方法を説明しています。

詳細については、「コレクションを定義する」を参照してください。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

/// <summary>
/// Uses the Amazon Rekognition Service to describe the contents of a
/// collection.
/// </summary>
public class DescribeCollection
{
    public static async Task Main()
    {
        var rekognitionClient = new AmazonRekognitionClient();

        string collectionId = "MyCollection";
        Console.WriteLine($"Describing collection: {collectionId}");
```

```
            var describeCollectionRequest = new DescribeCollectionRequest()
            {
                CollectionId = collectionId,
            };

            var describeCollectionResponse = await
rekognitionClient.DescribeCollectionAsync(describeCollectionRequest);
            Console.WriteLine($"Collection ARN:
{describeCollectionResponse.CollectionARN}");
            Console.WriteLine($"Face count:
{describeCollectionResponse.FaceCount}");
            Console.WriteLine($"Face model version:
{describeCollectionResponse.FaceModelVersion}");
            Console.WriteLine($"Created:
{describeCollectionResponse.CreationTimestamp}");
        }
    }
```

- API の詳細については、「 API リファレンスDescribeCollection」の「」を参照してください。 AWS SDK for .NET

`DetectFaces`

次の例は、`DetectFaces` を使用する方法を説明しています。

詳細については、「イメージ内の顔を検出する」を参照してください。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;
```

```
    /// <summary>
    /// Uses the Amazon Rekognition Service to detect faces within an image
    /// stored in an Amazon Simple Storage Service (Amazon S3) bucket.
    /// </summary>
    public class DetectFaces
    {
        public static async Task Main()
        {
            string photo = "input.jpg";
            string bucket = "bucket";

            var rekognitionClient = new AmazonRekognitionClient();

            var detectFacesRequest = new DetectFacesRequest()
            {
                Image = new Image()
                {
                    S3Object = new S3Object()
                    {
                        Name = photo,
                        Bucket = bucket,
                    },
                },

                // Attributes can be "ALL" or "DEFAULT".
                // "DEFAULT": BoundingBox, Confidence, Landmarks, Pose, and Quality.
                // "ALL": See https://docs.aws.amazon.com/sdkfornet/v3/apidocs/
items/Rekognition/TFaceDetail.html
                Attributes = new List<string>() { "ALL" },
            };

            try
            {
                DetectFacesResponse detectFacesResponse = await
 rekognitionClient.DetectFacesAsync(detectFacesRequest);
                bool hasAll = detectFacesRequest.Attributes.Contains("ALL");
                foreach (FaceDetail face in detectFacesResponse.FaceDetails)
                {
                    Console.WriteLine($"BoundingBox: top={face.BoundingBox.Left}
 left={face.BoundingBox.Top} width={face.BoundingBox.Width}
 height={face.BoundingBox.Height}");
                    Console.WriteLine($"Confidence: {face.Confidence}");
                    Console.WriteLine($"Landmarks: {face.Landmarks.Count}");
```

```
                    Console.WriteLine($"Pose: pitch={face.Pose.Pitch}
roll={face.Pose.Roll} yaw={face.Pose.Yaw}");
                    Console.WriteLine($"Brightness:
{face.Quality.Brightness}\tSharpness: {face.Quality.Sharpness}");

                    if (hasAll)
                    {
                        Console.WriteLine($"Estimated age is between
{face.AgeRange.Low} and {face.AgeRange.High} years old.");
                    }
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }
```

イメージ内のすべての顔の境界ボックス情報を表示します。

```
    using System;
    using System.Collections.Generic;
    using System.Drawing;
    using System.IO;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses the Amazon Rekognition Service to display the details of the
    /// bounding boxes around the faces detected in an image.
    /// </summary>
    public class ImageOrientationBoundingBox
    {
        public static async Task Main()
        {
            string photo = @"D:\Development\AWS-Examples\Rekognition\target.jpg"; //
"photo.jpg";

            var rekognitionClient = new AmazonRekognitionClient();
```

```
            var image = new Amazon.Rekognition.Model.Image();
            try
            {
                using var fs = new FileStream(photo, FileMode.Open,
FileAccess.Read);
                byte[] data = null;
                data = new byte[fs.Length];
                fs.Read(data, 0, (int)fs.Length);
                image.Bytes = new MemoryStream(data);
            }
            catch (Exception)
            {
                Console.WriteLine("Failed to load file " + photo);
                return;
            }

            int height;
            int width;

            // Used to extract original photo width/height
            using (var imageBitmap = new Bitmap(photo))
            {
                height = imageBitmap.Height;
                width = imageBitmap.Width;
            }

            Console.WriteLine("Image Information:");
            Console.WriteLine(photo);
            Console.WriteLine("Image Height: " + height);
            Console.WriteLine("Image Width: " + width);

            try
            {
                var detectFacesRequest = new DetectFacesRequest()
                {
                    Image = image,
                    Attributes = new List<string>() { "ALL" },
                };

                DetectFacesResponse detectFacesResponse = await
rekognitionClient.DetectFacesAsync(detectFacesRequest);
                detectFacesResponse.FaceDetails.ForEach(face =>
                {
```

```
                    Console.WriteLine("Face:");
                    ShowBoundingBoxPositions(
                        height,
                        width,
                        face.BoundingBox,
                        detectFacesResponse.OrientationCorrection);

                    Console.WriteLine($"BoundingBox: top={face.BoundingBox.Left}
 left={face.BoundingBox.Top} width={face.BoundingBox.Width}
 height={face.BoundingBox.Height}");
                    Console.WriteLine($"The detected face is estimated to be between
 {face.AgeRange.Low} and {face.AgeRange.High} years old.\n");
                });
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }

        /// <summary>
        /// Display the bounding box information for an image.
        /// </summary>
        /// <param name="imageHeight">The height of the image.</param>
        /// <param name="imageWidth">The width of the image.</param>
        /// <param name="box">The bounding box for a face found within the image.</
param>
        /// <param name="rotation">The rotation of the face's bounding box.</param>
        public static void ShowBoundingBoxPositions(int imageHeight, int imageWidth,
 BoundingBox box, string rotation)
        {
            float left;
            float top;

            if (rotation == null)
            {
                Console.WriteLine("No estimated orientation. Check Exif data.");
                return;
            }

            // Calculate face position based on image orientation.
            switch (rotation)
            {
                case "ROTATE_0":
```

```
                    left = imageWidth * box.Left;
                    top = imageHeight * box.Top;
                    break;
                case "ROTATE_90":
                    left = imageHeight * (1 - (box.Top + box.Height));
                    top = imageWidth * box.Left;
                    break;
                case "ROTATE_180":
                    left = imageWidth - (imageWidth * (box.Left + box.Width));
                    top = imageHeight * (1 - (box.Top + box.Height));
                    break;
                case "ROTATE_270":
                    left = imageHeight * box.Top;
                    top = imageWidth * (1 - box.Left - box.Width);
                    break;
                default:
                    Console.WriteLine("No estimated orientation information. Check
Exif data.");
                    return;
            }

            // Display face location information.
            Console.WriteLine($"Left: {left}");
            Console.WriteLine($"Top: {top}");
            Console.WriteLine($"Face Width: {imageWidth * box.Width}");
            Console.WriteLine($"Face Height: {imageHeight * box.Height}");
        }
    }
```

- API の詳細については、「 API リファレンスDetectFaces」の「」を参照してください。 AWS SDK for .NET

DetectLabels

次の例は、DetectLabels を使用する方法を説明しています。

詳細については、「イメージ内のラベルを検出する」を参照してください。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

/// <summary>
/// Uses the Amazon Rekognition Service to detect labels within an image
/// stored in an Amazon Simple Storage Service (Amazon S3) bucket.
/// </summary>
public class DetectLabels
{
    public static async Task Main()
    {
        string photo = "del_river_02092020_01.jpg"; // "input.jpg";
        string bucket = "igsmiths3photos"; // "bucket";

        var rekognitionClient = new AmazonRekognitionClient();

        var detectlabelsRequest = new DetectLabelsRequest
        {
            Image = new Image()
            {
                S3Object = new S3Object()
                {
                    Name = photo,
                    Bucket = bucket,
                },
            },
            MaxLabels = 10,
            MinConfidence = 75F,
        };

        try
        {
```

```
                DetectLabelsResponse detectLabelsResponse = await
rekognitionClient.DetectLabelsAsync(detectlabelsRequest);
                Console.WriteLine("Detected labels for " + photo);
                foreach (Label label in detectLabelsResponse.Labels)
                {
                    Console.WriteLine($"Name: {label.Name} Confidence:
{label.Confidence}");
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }
```

コンピュータに保存されているイメージファイル内のラベルを検出します。

```
    using System;
    using System.IO;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses the Amazon Rekognition Service to detect labels within an image
    /// stored locally.
    /// </summary>
    public class DetectLabelsLocalFile
    {
        public static async Task Main()
        {
            string photo = "input.jpg";

            var image = new Amazon.Rekognition.Model.Image();
            try
            {
                using var fs = new FileStream(photo, FileMode.Open,
FileAccess.Read);
                byte[] data = null;
                data = new byte[fs.Length];
```

```
                fs.Read(data, 0, (int)fs.Length);
                image.Bytes = new MemoryStream(data);
            }
            catch (Exception)
            {
                Console.WriteLine("Failed to load file " + photo);
                return;
            }

            var rekognitionClient = new AmazonRekognitionClient();

            var detectlabelsRequest = new DetectLabelsRequest
            {
                Image = image,
                MaxLabels = 10,
                MinConfidence = 77F,
            };

            try
            {
                DetectLabelsResponse detectLabelsResponse = await
rekognitionClient.DetectLabelsAsync(detectlabelsRequest);
                Console.WriteLine($"Detected labels for {photo}");
                foreach (Label label in detectLabelsResponse.Labels)
                {
                    Console.WriteLine($"{label.Name}: {label.Confidence}");
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }
```

- API の詳細については、「 API リファレンスDetectLabels」の「」を参照してください。 AWS SDK for .NET

DetectModerationLabels

次の例は、DetectModerationLabels を使用する方法を説明しています。

詳細については、「不適切なイメージを検出する」を参照してください。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

/// <summary>
/// Uses the Amazon Rekognition Service to detect unsafe content in a
/// JPEG or PNG format image.
/// </summary>
public class DetectModerationLabels
{
    public static async Task Main(string[] args)
    {
        string photo = "input.jpg";
        string bucket = "bucket";

        var rekognitionClient = new AmazonRekognitionClient();

        var detectModerationLabelsRequest = new DetectModerationLabelsRequest()
        {
            Image = new Image()
            {
                S3Object = new S3Object()
                {
                    Name = photo,
                    Bucket = bucket,
                },
            },
            MinConfidence = 60F,
        };

        try
```

```
            {
                var detectModerationLabelsResponse = await
rekognitionClient.DetectModerationLabelsAsync(detectModerationLabelsRequest);
                Console.WriteLine("Detected labels for " + photo);
                foreach (ModerationLabel label in
detectModerationLabelsResponse.ModerationLabels)
                {
                    Console.WriteLine($"Label: {label.Name}");
                    Console.WriteLine($"Confidence: {label.Confidence}");
                    Console.WriteLine($"Parent: {label.ParentName}");
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }
```

- API の詳細については、「 API リファレンスDetectModerationLabels」の「」を参照してください。 AWS SDK for .NET

`DetectText`

次の例は、`DetectText` を使用する方法を説明しています。

詳細については、「イメージ内のテキストを検出する」を参照してください。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

```
    /// <summary>
    /// Uses the Amazon Rekognition Service to detect text in an image. The
    /// example was created using the AWS SDK for .NET version 3.7 and .NET
    /// Core 5.0.
    /// </summary>
    public class DetectText
    {
        public static async Task Main()
        {
            string photo = "Dad_photographer.jpg"; // "input.jpg";
            string bucket = "igsmiths3photos"; // "bucket";

            var rekognitionClient = new AmazonRekognitionClient();

            var detectTextRequest = new DetectTextRequest()
            {
                Image = new Image()
                {
                    S3Object = new S3Object()
                    {
                        Name = photo,
                        Bucket = bucket,
                    },
                },
            };

            try
            {
                DetectTextResponse detectTextResponse = await
rekognitionClient.DetectTextAsync(detectTextRequest);
                Console.WriteLine($"Detected lines and words for {photo}");
                detectTextResponse.TextDetections.ForEach(text =>
                {
                    Console.WriteLine($"Detected: {text.DetectedText}");
                    Console.WriteLine($"Confidence: {text.Confidence}");
                    Console.WriteLine($"Id : {text.Id}");
                    Console.WriteLine($"Parent Id: {text.ParentId}");
                    Console.WriteLine($"Type: {text.Type}");
                });
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
```

```
            }
        }
    }
```

- API の詳細については、「 API リファレンスDetectText」の「」を参照してください。 AWS SDK for .NET

`GetCelebrityInfo`

次の例は、`GetCelebrityInfo` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

/// <summary>
/// Shows how to use Amazon Rekognition to retrieve information about the
/// celebrity identified by the supplied celebrity Id.
/// </summary>
public class CelebrityInfo
{
    public static async Task Main()
    {
        string celebId = "nnnnnnnn";

        var rekognitionClient = new AmazonRekognitionClient();

        var celebrityInfoRequest = new GetCelebrityInfoRequest
        {
            Id = celebId,
```

```
            };

            Console.WriteLine($"Getting information for celebrity: {celebId}");

            var celebrityInfoResponse = await
rekognitionClient.GetCelebrityInfoAsync(celebrityInfoRequest);

            // Display celebrity information.
            Console.WriteLine($"celebrity name: {celebrityInfoResponse.Name}");
            Console.WriteLine("Further information (if available):");
            celebrityInfoResponse.Urls.ForEach(url =>
            {
                Console.WriteLine(url);
            });
        }
    }
```

- API の詳細については、「 API リファレンスGetCelebrityInfo」の「」を参照してください。 AWS SDK for .NET

`IndexFaces`

次の例は、IndexFaces を使用する方法を説明しています。

詳細については、「コレクションに顔を追加する」を参照してください。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;
```

```
   /// <summary>
   /// Uses the Amazon Rekognition Service to detect faces in an image
   /// that has been uploaded to an Amazon Simple Storage Service (Amazon S3)
   /// bucket and then adds the information to a collection.
   /// </summary>
   public class AddFaces
   {
       public static async Task Main()
       {
           string collectionId = "MyCollection2";
           string bucket = "doc-example-bucket";
           string photo = "input.jpg";

           var rekognitionClient = new AmazonRekognitionClient();

           var image = new Image
           {
               S3Object = new S3Object
               {
                   Bucket = bucket,
                   Name = photo,
               },
           };

           var indexFacesRequest = new IndexFacesRequest
           {
               Image = image,
               CollectionId = collectionId,
               ExternalImageId = photo,
               DetectionAttributes = new List<string>() { "ALL" },
           };

           IndexFacesResponse indexFacesResponse = await
rekognitionClient.IndexFacesAsync(indexFacesRequest);

           Console.WriteLine($"{photo} added");
           foreach (FaceRecord faceRecord in indexFacesResponse.FaceRecords)
           {
               Console.WriteLine($"Face detected: Faceid is
{faceRecord.Face.FaceId}");
           }
       }
   }
```

- API の詳細については、「 API リファレンスIndexFaces」の「」を参照してください。 AWS SDK for .NET

`ListCollections`

次の例は、`ListCollections` を使用する方法を説明しています。

コレクションの詳細については、「コレクションを一覧表示する」を参照してください。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

/// <summary>
/// Uses Amazon Rekognition to list the collection IDs in the
/// current account.
/// </summary>
public class ListCollections
{
    public static async Task Main()
    {
        var rekognitionClient = new AmazonRekognitionClient();

        Console.WriteLine("Listing collections");
        int limit = 10;

        var listCollectionsRequest = new ListCollectionsRequest
        {
            MaxResults = limit,
        };
```

```
            var listCollectionsResponse = new ListCollectionsResponse();

            do
            {
                if (listCollectionsResponse is not null)
                {
                    listCollectionsRequest.NextToken =
listCollectionsResponse.NextToken;
                }

                listCollectionsResponse = await
rekognitionClient.ListCollectionsAsync(listCollectionsRequest);

                listCollectionsResponse.CollectionIds.ForEach(id =>
                {
                    Console.WriteLine(id);
                });
            }
            while (listCollectionsResponse.NextToken is not null);
        }
    }
```

- API の詳細については、「 API リファレンスListCollections」の「」を参照してください。 AWS SDK for .NET

`ListFaces`

次の例は、ListFaces を使用する方法を説明しています。

詳細については、「コレクションに顔を保存する」を参照してください。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    using System;
```

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

    /// <summary>
    /// Uses the Amazon Rekognition Service to retrieve the list of faces
    /// stored in a collection.
    /// </summary>
    public class ListFaces
    {
        public static async Task Main()
        {
            string collectionId = "MyCollection2";

            var rekognitionClient = new AmazonRekognitionClient();

            var listFacesResponse = new ListFacesResponse();
            Console.WriteLine($"Faces in collection {collectionId}");

            var listFacesRequest = new ListFacesRequest
            {
                CollectionId = collectionId,
                MaxResults = 1,
            };

            do
            {
                listFacesResponse = await
rekognitionClient.ListFacesAsync(listFacesRequest);
                listFacesResponse.Faces.ForEach(face =>
                {
                    Console.WriteLine(face.FaceId);
                });

                listFacesRequest.NextToken = listFacesResponse.NextToken;
            }
            while (!string.IsNullOrEmpty(listFacesResponse.NextToken));
        }
    }
```

- API の詳細については、「 API リファレンスListFaces」の「」を参照してください。 AWS SDK for .NET

`RecognizeCelebrities`

次の例は、RecognizeCelebrities を使用する方法を説明しています。

詳細については、「イメージ内で有名人を認識する」を参照してください。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    using System;
    using System.IO;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Shows how to use Amazon Rekognition to identify celebrities in a photo.
    /// </summary>
    public class CelebritiesInImage
    {
        public static async Task Main(string[] args)
        {
            string photo = "moviestars.jpg";

            var rekognitionClient = new AmazonRekognitionClient();

            var recognizeCelebritiesRequest = new RecognizeCelebritiesRequest();

            var img = new Amazon.Rekognition.Model.Image();
            byte[] data = null;
            try
            {
                using var fs = new FileStream(photo, FileMode.Open,
FileAccess.Read);
```

```
                data = new byte[fs.Length];
                fs.Read(data, 0, (int)fs.Length);
            }
            catch (Exception)
            {
                Console.WriteLine($"Failed to load file {photo}");
                return;
            }

            img.Bytes = new MemoryStream(data);
            recognizeCelebritiesRequest.Image = img;

            Console.WriteLine($"Looking for celebrities in image {photo}\n");

            var recognizeCelebritiesResponse = await
rekognitionClient.RecognizeCelebritiesAsync(recognizeCelebritiesRequest);

            Console.WriteLine($"{recognizeCelebritiesResponse.CelebrityFaces.Count}
celebrity(s) were recognized.\n");
            recognizeCelebritiesResponse.CelebrityFaces.ForEach(celeb =>
            {
                Console.WriteLine($"Celebrity recognized: {celeb.Name}");
                Console.WriteLine($"Celebrity ID: {celeb.Id}");
                BoundingBox boundingBox = celeb.Face.BoundingBox;
                Console.WriteLine($"position: {boundingBox.Left}
{boundingBox.Top}");
                Console.WriteLine("Further information (if available):");
                celeb.Urls.ForEach(url =>
                {
                    Console.WriteLine(url);
                });
            });


Console.WriteLine($"{recognizeCelebritiesResponse.UnrecognizedFaces.Count} face(s)
were unrecognized.");
        }
    }
```

- API の詳細については、「 API リファレンスRecognizeCelebrities」の「」を参照してください。 AWS SDK for .NET

SearchFaces

次の例は、SearchFaces を使用する方法を説明しています。

詳細については、顔 (フェイス ID) を検索する を参照してください。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

    /// <summary>
    /// Uses the Amazon Rekognition Service to find faces in an image that
    /// match the face Id provided in the method request.
    /// </summary>
    public class SearchFacesMatchingId
    {
        public static async Task Main()
        {
            string collectionId = "MyCollection";
            string faceId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";

            var rekognitionClient = new AmazonRekognitionClient();

            // Search collection for faces matching the face id.
            var searchFacesRequest = new SearchFacesRequest
            {
                CollectionId = collectionId,
                FaceId = faceId,
                FaceMatchThreshold = 70F,
                MaxFaces = 2,
            };

            SearchFacesResponse searchFacesResponse = await
rekognitionClient.SearchFacesAsync(searchFacesRequest);
```

```
            Console.WriteLine("Face matching faceId " + faceId);

            Console.WriteLine("Matche(s): ");
            searchFacesResponse.FaceMatches.ForEach(face =>
            {
                Console.WriteLine($"FaceId: {face.Face.FaceId} Similarity:
{face.Similarity}");
            });
        }
    }
```

- API の詳細については、「 API リファレンスSearchFaces」の「」を参照してください。 AWS SDK for .NET

`SearchFacesByImage`

次の例は、SearchFacesByImage を使用する方法を説明しています。

詳細については、「顔を検索する (イメージ)」を参照してください。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

    /// <summary>
    /// Uses the Amazon Rekognition Service to search for images matching those
    /// in a collection.
    /// </summary>
    public class SearchFacesMatchingImage
```

```
    {
        public static async Task Main()
        {
            string collectionId = "MyCollection";
            string bucket = "bucket";
            string photo = "input.jpg";

            var rekognitionClient = new AmazonRekognitionClient();

            // Get an image object from S3 bucket.
            var image = new Image()
            {
                S3Object = new S3Object()
                {
                    Bucket = bucket,
                    Name = photo,
                },
            };

            var searchFacesByImageRequest = new SearchFacesByImageRequest()
            {
                CollectionId = collectionId,
                Image = image,
                FaceMatchThreshold = 70F,
                MaxFaces = 2,
            };

            SearchFacesByImageResponse searchFacesByImageResponse = await
rekognitionClient.SearchFacesByImageAsync(searchFacesByImageRequest);

            Console.WriteLine("Faces matching largest face in image from " + photo);
            searchFacesByImageResponse.FaceMatches.ForEach(face =>
            {
                Console.WriteLine($"FaceId: {face.Face.FaceId}, Similarity:
{face.Similarity}");
            });
        }
    }
```

- API の詳細については、「 API リファレンスSearchFacesByImage」の「」を参照してください。 AWS SDK for .NET

を使用した Route 53 ドメイン登録の例 AWS SDK for .NET

次のコード例は、Route 53 ドメイン登録 AWS SDK for .NET で 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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。ここでは GitHub、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。

開始方法

ハロー Route 53 ドメイン登録

以下のコード例は、Route 53 ドメイン登録の使用を開始する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
public static class HelloRoute53Domains
{
    static async Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
 the Amazon Route 53 domain registration 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<IAmazonRoute53Domains>()
            ).Build();
```

```
        // Now the client is available for injection.
        var route53Client =
host.Services.GetRequiredService<IAmazonRoute53Domains>();

        // You can use await and any of the async methods to get a response.
        var response = await route53Client.ListPricesAsync(new ListPricesRequest
{ Tld = "com" });
        Console.WriteLine($"Hello Amazon Route 53 Domains! Following are prices
for .com domain operations:");
        var comPrices = response.Prices.FirstOrDefault();
        if (comPrices != null)
        {
            Console.WriteLine($"\tRegistration: {comPrices.RegistrationPrice?.Price}
{comPrices.RegistrationPrice?.Currency}");
            Console.WriteLine($"\tRenewal: {comPrices.RenewalPrice?.Price}
{comPrices.RenewalPrice?.Currency}");
        }
    }
}
```

- API の詳細については、「 API リファレンスListPrices」の「」を参照してください。 AWS SDK for .NET

トピック

- アクション
- シナリオ

アクション

`CheckDomainAvailability`

次の例は、`CheckDomainAvailability` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
/// Check the availability of a domain name.
/// </summary>
/// <param name="domain">The domain to check for availability.</param>
/// <returns>An availability result string.</returns>
public async Task<string> CheckDomainAvailability(string domain)
{
    var result = await _amazonRoute53Domains.CheckDomainAvailabilityAsync(
        new CheckDomainAvailabilityRequest
        {
            DomainName = domain
        }
    );
    return result.Availability.Value;
}
```

- API の詳細については、「 API リファレンスCheckDomainAvailability」の「」を参照してください。 AWS SDK for .NET

`CheckDomainTransferability`

次の例は、CheckDomainTransferability を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
/// Check the transferability of a domain name.
/// </summary>
/// <param name="domain">The domain to check for transferability.</param>
/// <returns>A transferability result string.</returns>
public async Task<string> CheckDomainTransferability(string domain)
{
```

```
        var result = await _amazonRoute53Domains.CheckDomainTransferabilityAsync(
            new CheckDomainTransferabilityRequest
            {
                DomainName = domain
            }
        );
        return result.Transferability.Transferable.Value;
    }
```

- API の詳細については、「 API リファレンスCheckDomainTransferability」の「」を参照してください。 AWS SDK for .NET

`GetDomainDetail`

次の例は、GetDomainDetail を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
/// Get details for a domain.
/// </summary>
/// <returns>A string with detail information about the domain.</returns>
public async Task<string> GetDomainDetail(string domainName)
{
    try
    {
        var result = await _amazonRoute53Domains.GetDomainDetailAsync(
            new GetDomainDetailRequest()
            {
                DomainName = domainName
            });
        var details = $"\tDomain {domainName}:\n" +
```

```
                        $"\tCreated on {result.CreationDate.ToShortDateString()}.
\n" +
                        $"\tAdmin contact is {result.AdminContact.Email}.\n" +
                        $"\tAuto-renew is {result.AutoRenew}.\n";

            return details;
        }
        catch (InvalidInputException)
        {
            return $"Domain {domainName} was not found in your account.";
        }
    }
```

- API の詳細については、「 API リファレンスGetDomainDetail」の「」を参照してください。AWS SDK for .NET

`GetDomainSuggestions`

次の例は、GetDomainSuggestions を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get a list of suggestions for a given domain.
    /// </summary>
    /// <param name="domain">The domain to check for suggestions.</param>
    /// <param name="onlyAvailable">If true, only returns available domains.</param>
    /// <param name="suggestionCount">The number of suggestions to return. Defaults
to the max of 50.</param>
    /// <returns>A collection of domain suggestions.</returns>
    public async Task<List<DomainSuggestion>> GetDomainSuggestions(string domain,
bool onlyAvailable, int suggestionCount = 50)
    {
```

```
        var result = await _amazonRoute53Domains.GetDomainSuggestionsAsync(
            new GetDomainSuggestionsRequest
            {
                DomainName = domain,
                OnlyAvailable = onlyAvailable,
                SuggestionCount = suggestionCount
            }
        );
        return result.SuggestionsList;
    }
```

- API の詳細については、「 API リファレンスGetDomainSuggestions」の「」を参照してください。 AWS SDK for .NET

`GetOperationDetail`

次の例は、`GetOperationDetail` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get details for a domain action operation.
    /// </summary>
    /// <param name="operationId">The operational Id.</param>
    /// <returns>A string describing the operational details.</returns>
    public async Task<string> GetOperationDetail(string? operationId)
    {
        if (operationId == null)
            return "Unable to get operational details because ID is null.";
        try
        {
            var operationDetails =
                await _amazonRoute53Domains.GetOperationDetailAsync(
```

```
                    new GetOperationDetailRequest
                    {
                        OperationId = operationId
                    }
                );

            var details = $"\tOperation {operationId}:\n" +
                          $"\tFor domain {operationDetails.DomainName} on
{operationDetails.SubmittedDate.ToShortDateString()}.\n" +
                          $"\tMessage is {operationDetails.Message}.\n" +
                          $"\tStatus is {operationDetails.Status}.\n";

            return details;
        }
        catch (AmazonRoute53DomainsException ex)
        {
            return $"Unable to get operation details. Here's why: {ex.Message}.";
        }
    }
```

- API の詳細については、「 API リファレンスGetOperationDetail」の「」を参照してください。 AWS SDK for .NET

`ListDomains`

次の例は、ListDomains を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// List the domains for the account.
    /// </summary>
    /// <returns>A collection of domain summary records.</returns>
```

```
    public async Task<List<DomainSummary>> ListDomains()
    {
        var results = new List<DomainSummary>();
        var paginateDomains = _amazonRoute53Domains.Paginators.ListDomains(
            new ListDomainsRequest());

        // Get the entire list using the paginator.
        await foreach (var domain in paginateDomains.Domains)
        {
            results.Add(domain);
        }
        return results;
    }
```

- API の詳細については、「 API リファレンスListDomains」の「」を参照してください。 AWS SDK for .NET

`ListOperations`

次の例は、ListOperations を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// List operations for the account that are submitted after a specified date.
    /// </summary>
    /// <returns>A collection of operation summary records.</returns>
    public async Task<List<OperationSummary>> ListOperations(DateTime
submittedSince)
    {
        var results = new List<OperationSummary>();
        var paginateOperations = _amazonRoute53Domains.Paginators.ListOperations(
            new ListOperationsRequest()
```

```
            {
                SubmittedSince = submittedSince
            });

        // Get the entire list using the paginator.
        await foreach (var operations in paginateOperations.Operations)
        {
            results.Add(operations);
        }
        return results;
    }
```

- API の詳細については、「 API リファレンスListOperations」の「」を参照してください。AWS SDK for .NET

`ListPrices`

次の例は、ListPrices を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// List prices for domain type operations.
    /// </summary>
    /// <param name="domainTypes">Domain types to include in the results.</param>
    /// <returns>The list of domain prices.</returns>
    public async Task<List<DomainPrice>> ListPrices(List<string> domainTypes)
    {
        var results = new List<DomainPrice>();
        var paginatePrices = _amazonRoute53Domains.Paginators.ListPrices(new
ListPricesRequest());
        // Get the entire list using the paginator.
        await foreach (var prices in paginatePrices.Prices)
```

```
        {
            results.Add(prices);
        }
        return results.Where(p => domainTypes.Contains(p.Name)).ToList();
    }
```

- API の詳細については、「 API リファレンスListPrices」の「」を参照してください。 AWS SDK for .NET

`RegisterDomain`

次の例は、RegisterDomain を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Initiate a domain registration request.
    /// </summary>
    /// <param name="contact">Contact details.</param>
    /// <param name="domainName">The domain name to register.</param>
    /// <param name="autoRenew">True if the domain should automatically renew.</
param>
    /// <param name="duration">The duration in years for the domain registration.</
param>
    /// <returns>The operation Id.</returns>
    public async Task<string?> RegisterDomain(string domainName, bool autoRenew, int
 duration, ContactDetail contact)
    {
        // This example uses the same contact information for admin, registrant, and
 tech contacts.
        try
        {
            var result = await _amazonRoute53Domains.RegisterDomainAsync(
```

```
                new RegisterDomainRequest()
                {
                    AdminContact = contact,
                    RegistrantContact = contact,
                    TechContact = contact,
                    DomainName = domainName,
                    AutoRenew = autoRenew,
                    DurationInYears = duration,
                    PrivacyProtectAdminContact = false,
                    PrivacyProtectRegistrantContact = false,
                    PrivacyProtectTechContact = false
                }
            );
            return result.OperationId;
        }
        catch (InvalidInputException)
        {
            _logger.LogInformation($"Unable to request registration for domain
{domainName}");
            return null;
        }
    }
```

- API の詳細については、「 API リファレンスRegisterDomain」の「」を参照してください。AWS SDK for .NET

`ViewBilling`

次の例は、`ViewBilling` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
```

```
    /// View billing records for the account between a start and end date.
    /// </summary>
    /// <param name="startDate">The start date for billing results.</param>
    /// <param name="endDate">The end date for billing results.</param>
    /// <returns>A collection of billing records.</returns>
    public async Task<List<BillingRecord>> ViewBilling(DateTime startDate, DateTime
endDate)
    {
        var results = new List<BillingRecord>();
        var paginateBilling = _amazonRoute53Domains.Paginators.ViewBilling(
            new ViewBillingRequest()
            {
                Start = startDate,
                End = endDate
            });

        // Get the entire list using the paginator.
        await foreach (var billingRecords in paginateBilling.BillingRecords)
        {
            results.Add(billingRecords);
        }
        return results;
    }
```

- API の詳細については、「 API リファレンスViewBilling」の「」を参照してください。 AWS SDK for .NET

シナリオ

ドメインを始める

次のコードサンプルは、以下の操作方法を示しています。

- 現在のドメインを一覧表示し、過去 1 年間の操作を一覧表示します。
- 過去 1 年間の請求記録とドメインタイプの価格を表示します。
- ドメインの候補を取得します。
- ドメインの可用性と移管可能性を確認します。
- オプションで、ドメイン登録をリクエストします。
- 操作の詳細を入手します。

- オプションで、ドメインの詳細を取得します。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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のシナリオを実行します。

```
public static class Route53DomainScenario
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.

    This .NET example performs the following tasks:
        1. List current domains.
        2. List operations in the past year.
        3. View billing for the account in the past year.
        4. View prices for domain types.
        5. Get domain suggestions.
        6. Check domain availability.
        7. Check domain transferability.
        8. Optionally, request a domain registration.
        9. Get an operation detail.
       10. Optionally, get a domain detail.
   */

    private static Route53Wrapper _route53Wrapper = null!;
    private static IConfiguration _configuration = 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<IAmazonRoute53Domains>()
                .AddTransient<Route53Wrapper>()
            )
            .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();

        var logger = LoggerFactory.Create(builder =>
        {
            builder.AddConsole();
        }).CreateLogger(typeof(Route53DomainScenario));

        _route53Wrapper = host.Services.GetRequiredService<Route53Wrapper>();

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the Amazon Route 53 domains example
scenario.");
        Console.WriteLine(new string('-', 80));

        try
        {
            await ListDomains();
            await ListOperations();
            await ListBillingRecords();
            await ListPrices();
            await ListDomainSuggestions();
            await CheckDomainAvailability();
            await CheckDomainTransferability();
            var operationId = await RequestDomainRegistration();
            await GetOperationalDetail(operationId);
            await GetDomainDetails();
        }
        catch (Exception ex)
        {
            logger.LogError(ex, "There was a problem executing the scenario.");
```

```
        }

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("The Amazon Route 53 domains example scenario is
complete.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List account registered domains.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task ListDomains()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"1. List account domains.");
        var domains = await _route53Wrapper.ListDomains();
        for (int i = 0; i < domains.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {domains[i].DomainName}");
        }

        if (!domains.Any())
        {
            Console.WriteLine("\tNo domains found in this account.");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List domain operations in the past year.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task ListOperations()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"2. List account domain operations in the past year.");
        var operations = await _route53Wrapper.ListOperations(
            DateTime.Today.AddYears(-1));
        for (int i = 0; i < operations.Count; i++)
        {
            Console.WriteLine($"\tOperation Id: {operations[i].OperationId}");
            Console.WriteLine($"\tStatus: {operations[i].Status}");
```

```
            Console.WriteLine($"\tDate: {operations[i].SubmittedDate}");
        }
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List billing in the past year.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task ListBillingRecords()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"3. View billing for the account in the past year.");
        var billingRecords = await _route53Wrapper.ViewBilling(
            DateTime.Today.AddYears(-1),
            DateTime.Today);
        for (int i = 0; i < billingRecords.Count; i++)
        {
            Console.WriteLine($"\tBill Date:
{billingRecords[i].BillDate.ToShortDateString()}");
            Console.WriteLine($"\tOperation: {billingRecords[i].Operation}");
            Console.WriteLine($"\tPrice: {billingRecords[i].Price}");
        }
        if (!billingRecords.Any())
        {
            Console.WriteLine("\tNo billing records found in this account for the
past year.");
        }
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List prices for a few domain types.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task ListPrices()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"4. View prices for domain types.");
        var domainTypes = new List<string> { "net", "com", "org", "co" };

        var prices = await _route53Wrapper.ListPrices(domainTypes);
        foreach (var pr in prices)
        {
```

```
            Console.WriteLine($"\tName: {pr.Name}");
            Console.WriteLine($"\tRegistration: {pr.RegistrationPrice?.Price}
{pr.RegistrationPrice?.Currency}");
            Console.WriteLine($"\tRenewal: {pr.RenewalPrice?.Price}
{pr.RenewalPrice?.Currency}");
            Console.WriteLine($"\tTransfer: {pr.TransferPrice?.Price}
{pr.TransferPrice?.Currency}");
            Console.WriteLine($"\tChange Ownership: {pr.ChangeOwnershipPrice?.Price}
{pr.ChangeOwnershipPrice?.Currency}");
            Console.WriteLine($"\tRestoration: {pr.RestorationPrice?.Price}
{pr.RestorationPrice?.Currency}");
            Console.WriteLine();
        }
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List domain suggestions for a domain name.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task ListDomainSuggestions()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"5. Get domain suggestions.");
        string? domainName = null;
        while (domainName == null || string.IsNullOrWhiteSpace(domainName))
        {
            Console.WriteLine($"Enter a domain name to get available domain
suggestions.");
            domainName = Console.ReadLine();
        }

        var suggestions = await _route53Wrapper.GetDomainSuggestions(domainName,
true, 5);
        foreach (var suggestion in suggestions)
        {
            Console.WriteLine($"\tSuggestion Name: {suggestion.DomainName}");
            Console.WriteLine($"\tAvailability: {suggestion.Availability}");
        }
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Check availability for a domain name.
```

```
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CheckDomainAvailability()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"6. Check domain availability.");
        string? domainName = null;
        while (domainName == null || string.IsNullOrWhiteSpace(domainName))
        {
            Console.WriteLine($"Enter a domain name to check domain availability.");
            domainName = Console.ReadLine();
        }

        var availability = await
_route53Wrapper.CheckDomainAvailability(domainName);
        Console.WriteLine($"\tAvailability: {availability}");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Check transferability for a domain name.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CheckDomainTransferability()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"7. Check domain transferability.");
        string? domainName = null;
        while (domainName == null || string.IsNullOrWhiteSpace(domainName))
        {
            Console.WriteLine($"Enter a domain name to check domain
transferability.");
            domainName = Console.ReadLine();
        }

        var transferability = await
_route53Wrapper.CheckDomainTransferability(domainName);
        Console.WriteLine($"\tTransferability: {transferability}");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Check transferability for a domain name.
```

```
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task<string?> RequestDomainRegistration()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"8. Optionally, request a domain registration.");

        Console.WriteLine($"\tNote: This example uses domain request settings in
 settings.json.");
        Console.WriteLine($"\tTo change the domain registration settings, set the
 values in that file.");
        Console.WriteLine($"\tRemember, registering an actual domain will incur an
 account billing cost.");
        Console.WriteLine($"\tWould you like to begin a domain registration? (y/
n)");
        var ynResponse = Console.ReadLine();
        if (ynResponse != null && ynResponse.Equals("y",
 StringComparison.InvariantCultureIgnoreCase))
        {
            string domainName = _configuration["DomainName"];
            ContactDetail contact = new ContactDetail();
            contact.CountryCode =
 CountryCode.FindValue(_configuration["Contact:CountryCode"]);
            contact.ContactType =
 ContactType.FindValue(_configuration["Contact:ContactType"]);

            _configuration.GetSection("Contact").Bind(contact);

            var operationId = await _route53Wrapper.RegisterDomain(
                domainName,
                Convert.ToBoolean(_configuration["AutoRenew"]),
                Convert.ToInt32(_configuration["DurationInYears"]),
                contact);
            if (operationId != null)
            {
                Console.WriteLine(
                    $"\tRegistration requested. Operation Id: {operationId}");
            }

            return operationId;
        }

        Console.WriteLine(new string('-', 80));
        return null;
```

```
    }

    /// <summary>
    /// Get details for an operation.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetOperationalDetail(string? operationId)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"9. Get an operation detail.");

        var operationDetails =
            await _route53Wrapper.GetOperationDetail(operationId);

        Console.WriteLine(operationDetails);

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Optionally, get details for a registered domain.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task<string?> GetDomainDetails()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"10. Get details on a domain.");

        Console.WriteLine($"\tNote: you must have a registered domain to get
details.");
        Console.WriteLine($"\tWould you like to get domain details? (y/n)");
        var ynResponse = Console.ReadLine();
        if (ynResponse != null && ynResponse.Equals("y",
StringComparison.InvariantCultureIgnoreCase))
        {
            string? domainName = null;
            while (domainName == null)
            {
                Console.WriteLine($"\tEnter a domain name to get details.");
                domainName = Console.ReadLine();
            }

            var domainDetails = await _route53Wrapper.GetDomainDetail(domainName);
            Console.WriteLine(domainDetails);
```

```
        }

        Console.WriteLine(new string('-', 80));
        return null;
    }
}
```

Route 53 のドメイン登録アクションにシナリオが使用するラッパーメソッド。

```
public class Route53Wrapper
{
    private readonly IAmazonRoute53Domains _amazonRoute53Domains;
    private readonly ILogger<Route53Wrapper> _logger;
    public Route53Wrapper(IAmazonRoute53Domains amazonRoute53Domains,
 ILogger<Route53Wrapper> logger)
    {
        _amazonRoute53Domains = amazonRoute53Domains;
        _logger = logger;
    }


    /// <summary>
    /// List prices for domain type operations.
    /// </summary>
    /// <param name="domainTypes">Domain types to include in the results.</param>
    /// <returns>The list of domain prices.</returns>
    public async Task<List<DomainPrice>> ListPrices(List<string> domainTypes)
    {
        var results = new List<DomainPrice>();
        var paginatePrices = _amazonRoute53Domains.Paginators.ListPrices(new
 ListPricesRequest());
        // Get the entire list using the paginator.
        await foreach (var prices in paginatePrices.Prices)
        {
            results.Add(prices);
        }
        return results.Where(p => domainTypes.Contains(p.Name)).ToList();
    }


    /// <summary>
```

```
    /// Check the availability of a domain name.
    /// </summary>
    /// <param name="domain">The domain to check for availability.</param>
    /// <returns>An availability result string.</returns>
    public async Task<string> CheckDomainAvailability(string domain)
    {
        var result = await _amazonRoute53Domains.CheckDomainAvailabilityAsync(
            new CheckDomainAvailabilityRequest
            {
                DomainName = domain
            }
        );
        return result.Availability.Value;
    }


    /// <summary>
    /// Check the transferability of a domain name.
    /// </summary>
    /// <param name="domain">The domain to check for transferability.</param>
    /// <returns>A transferability result string.</returns>
    public async Task<string> CheckDomainTransferability(string domain)
    {
        var result = await _amazonRoute53Domains.CheckDomainTransferabilityAsync(
            new CheckDomainTransferabilityRequest
            {
                DomainName = domain
            }
        );
        return result.Transferability.Transferable.Value;
    }


    /// <summary>
    /// Get a list of suggestions for a given domain.
    /// </summary>
    /// <param name="domain">The domain to check for suggestions.</param>
    /// <param name="onlyAvailable">If true, only returns available domains.</param>
    /// <param name="suggestionCount">The number of suggestions to return. Defaults
to the max of 50.</param>
    /// <returns>A collection of domain suggestions.</returns>
    public async Task<List<DomainSuggestion>> GetDomainSuggestions(string domain,
bool onlyAvailable, int suggestionCount = 50)
    {
```

```
        var result = await _amazonRoute53Domains.GetDomainSuggestionsAsync(
            new GetDomainSuggestionsRequest
            {
                DomainName = domain,
                OnlyAvailable = onlyAvailable,
                SuggestionCount = suggestionCount
            }
        );
        return result.SuggestionsList;
    }


    /// <summary>
    /// Get details for a domain action operation.
    /// </summary>
    /// <param name="operationId">The operational Id.</param>
    /// <returns>A string describing the operational details.</returns>
    public async Task<string> GetOperationDetail(string? operationId)
    {
        if (operationId == null)
            return "Unable to get operational details because ID is null.";
        try
        {
            var operationDetails =
                await _amazonRoute53Domains.GetOperationDetailAsync(
                    new GetOperationDetailRequest
                    {
                        OperationId = operationId
                    }
                );

            var details = $"\tOperation {operationId}:\n" +
                          $"\tFor domain {operationDetails.DomainName} on
{operationDetails.SubmittedDate.ToShortDateString()}.\n" +
                          $"\tMessage is {operationDetails.Message}.\n" +
                          $"\tStatus is {operationDetails.Status}.\n";

            return details;
        }
        catch (AmazonRoute53DomainsException ex)
        {
            return $"Unable to get operation details. Here's why: {ex.Message}.";
        }
    }
```

```
    /// <summary>
    /// Initiate a domain registration request.
    /// </summary>
    /// <param name="contact">Contact details.</param>
    /// <param name="domainName">The domain name to register.</param>
    /// <param name="autoRenew">True if the domain should automatically renew.</
param>
    /// <param name="duration">The duration in years for the domain registration.</
param>
    /// <returns>The operation Id.</returns>
    public async Task<string?> RegisterDomain(string domainName, bool autoRenew, int
 duration, ContactDetail contact)
    {
        // This example uses the same contact information for admin, registrant, and
 tech contacts.
        try
        {
            var result = await _amazonRoute53Domains.RegisterDomainAsync(
                new RegisterDomainRequest()
                {
                    AdminContact = contact,
                    RegistrantContact = contact,
                    TechContact = contact,
                    DomainName = domainName,
                    AutoRenew = autoRenew,
                    DurationInYears = duration,
                    PrivacyProtectAdminContact = false,
                    PrivacyProtectRegistrantContact = false,
                    PrivacyProtectTechContact = false
                }
            );
            return result.OperationId;
        }
        catch (InvalidInputException)
        {
            _logger.LogInformation($"Unable to request registration for domain
 {domainName}");
            return null;
        }
    }
```

```
    /// <summary>
    /// View billing records for the account between a start and end date.
    /// </summary>
    /// <param name="startDate">The start date for billing results.</param>
    /// <param name="endDate">The end date for billing results.</param>
    /// <returns>A collection of billing records.</returns>
    public async Task<List<BillingRecord>> ViewBilling(DateTime startDate, DateTime
endDate)
    {
        var results = new List<BillingRecord>();
        var paginateBilling = _amazonRoute53Domains.Paginators.ViewBilling(
            new ViewBillingRequest()
            {
                Start = startDate,
                End = endDate
            });

        // Get the entire list using the paginator.
        await foreach (var billingRecords in paginateBilling.BillingRecords)
        {
            results.Add(billingRecords);
        }
        return results;
    }


    /// <summary>
    /// List the domains for the account.
    /// </summary>
    /// <returns>A collection of domain summary records.</returns>
    public async Task<List<DomainSummary>> ListDomains()
    {
        var results = new List<DomainSummary>();
        var paginateDomains = _amazonRoute53Domains.Paginators.ListDomains(
            new ListDomainsRequest());

        // Get the entire list using the paginator.
        await foreach (var domain in paginateDomains.Domains)
        {
            results.Add(domain);
        }
        return results;
    }
```

```
    /// <summary>
    /// List operations for the account that are submitted after a specified date.
    /// </summary>
    /// <returns>A collection of operation summary records.</returns>
    public async Task<List<OperationSummary>> ListOperations(DateTime
 submittedSince)
    {
        var results = new List<OperationSummary>();
        var paginateOperations = _amazonRoute53Domains.Paginators.ListOperations(
            new ListOperationsRequest()
            {
                SubmittedSince = submittedSince
            });

        // Get the entire list using the paginator.
        await foreach (var operations in paginateOperations.Operations)
        {
            results.Add(operations);
        }
        return results;
    }


    /// <summary>
    /// Get details for a domain.
    /// </summary>
    /// <returns>A string with detail information about the domain.</returns>
    public async Task<string> GetDomainDetail(string domainName)
    {
        try
        {
            var result = await _amazonRoute53Domains.GetDomainDetailAsync(
                new GetDomainDetailRequest()
                {
                    DomainName = domainName
                });
            var details = $"\tDomain {domainName}:\n" +
                          $"\tCreated on {result.CreationDate.ToShortDateString()}.
\n" +
                          $"\tAdmin contact is {result.AdminContact.Email}.\n" +
                          $"\tAuto-renew is {result.AutoRenew}.\n";

            return details;
```

```
        }
        catch (InvalidInputException)
        {
            return $"Domain {domainName} was not found in your account.";
        }
    }
}
```

- 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。
  - CheckDomainAvailability
  - CheckDomainTransferability
  - GetDomainDetail
  - GetDomainSuggestions
  - GetOperationDetail
  - ListDomains
  - ListOperations
  - ListPrices
  - RegisterDomain
  - ViewBilling

を使用した Amazon S3 の例 AWS SDK for .NET

次のコード例は、Amazon S3 AWS SDK for .NET で 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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。ここでは GitHub、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

- アクション
- シナリオ
- サーバーレスサンプル

アクション

`AbortMultipartUploads`

次の例は、AbortMultipartUploads を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
using System;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Transfer;

/// <summary>
/// This example shows how to use the Amazon Simple Storage Service
/// (Amazon S3) to stop a multi-part upload process using the Amazon S3
/// TransferUtility.
/// </summary>
public class AbortMPU
{
    public static async Task Main()
    {
        string bucketName = "doc-example-bucket";

        // If the AWS Region defined for your default user is different
        // from the Region where your Amazon S3 bucket is located,
        // pass the Region name to the S3 client object's constructor.
        // For example: RegionEndpoint.USWest2.
        IAmazonS3 client = new AmazonS3Client();
```

```
            await AbortMPUAsync(client, bucketName);
        }

        /// <summary>
        /// Cancels the multi-part copy process.
        /// </summary>
        /// <param name="client">The initialized client object used to create
        /// the TransferUtility object.</param>
        /// <param name="bucketName">The name of the S3 bucket where the
        /// multi-part copy operation is in progress.</param>
        public static async Task AbortMPUAsync(IAmazonS3 client, string bucketName)
        {
            try
            {
                var transferUtility = new TransferUtility(client);

                // Cancel all in-progress uploads initiated before the specified
date.
                await transferUtility.AbortMultipartUploadsAsync(
                    bucketName, DateTime.Now.AddDays(-7));
            }
            catch (AmazonS3Exception e)
            {
                Console.WriteLine($"Error: {e.Message}");
            }
        }
    }
```

- API の詳細については、「 API リファレンスAbortMultipartUploads」の「」を参照してください。 AWS SDK for .NET

CopyObject

次の例は、CopyObject を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    using System;
    using System.Threading.Tasks;
    using Amazon.S3;
    using Amazon.S3.Model;

    public class CopyObject
    {
        public static async Task Main()
        {
            // Specify the AWS Region where your buckets are located if it is
            // different from the AWS Region of the default user.
            IAmazonS3 s3Client = new AmazonS3Client();

            // Remember to change these values to refer to your Amazon S3 objects.
            string sourceBucketName = "doc-example-bucket1";
            string destinationBucketName = "doc-example-bucket2";
            string sourceObjectKey = "testfile.txt";
            string destinationObjectKey = "testfilecopy.txt";

            Console.WriteLine($"Copying {sourceObjectKey} from {sourceBucketName} to
");
            Console.WriteLine($"{destinationBucketName} as {destinationObjectKey}");

            var response = await CopyingObjectAsync(
                s3Client,
                sourceObjectKey,
                destinationObjectKey,
                sourceBucketName,
                destinationBucketName);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine("\nCopy complete.");
            }
```

```
        }

        /// <summary>
        /// This method calls the AWS SDK for .NET to copy an
        /// object from one Amazon S3 bucket to another.
        /// </summary>
        /// <param name="client">The Amazon S3 client object.</param>
        /// <param name="sourceKey">The name of the object to be copied.</param>
        /// <param name="destinationKey">The name under which to save the copy.</
param>
        /// <param name="sourceBucketName">The name of the Amazon S3 bucket
        /// where the file is located now.</param>
        /// <param name="destinationBucketName">The name of the Amazon S3
        /// bucket where the copy should be saved.</param>
        /// <returns>Returns a CopyObjectResponse object with the results from
        /// the async call.</returns>
        public static async Task<CopyObjectResponse> CopyingObjectAsync(
            IAmazonS3 client,
            string sourceKey,
            string destinationKey,
            string sourceBucketName,
            string destinationBucketName)
        {
            var response = new CopyObjectResponse();
            try
            {
                var request = new CopyObjectRequest
                {
                    SourceBucket = sourceBucketName,
                    SourceKey = sourceKey,
                    DestinationBucket = destinationBucketName,
                    DestinationKey = destinationKey,
                };
                response = await client.CopyObjectAsync(request);
            }
            catch (AmazonS3Exception ex)
            {
                Console.WriteLine($"Error copying object: '{ex.Message}'");
            }

            return response;
        }
    }
```

- API の詳細については、「 API リファレンスCopyObject」の「」を参照してください。 AWS SDK for .NET

`CreateBucket`

次の例は、`CreateBucket` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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="client">An initialized Amazon S3 client object.</param>
        /// <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 static async Task<bool> CreateBucketAsync(IAmazonS3 client, string
bucketName)
        {
            try
            {
                var request = new PutBucketRequest
                {
                    BucketName = bucketName,
                    UseClientRegion = true,
                };

                var response = await client.PutBucketAsync(request);
                return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
            }
            catch (AmazonS3Exception ex)
            {
```

```
                Console.WriteLine($"Error creating bucket: '{ex.Message}'");
                return false;
            }
        }
```

オブジェクトロックを有効にしてバケットを作成します。

```
    /// <summary>
    /// Create a new Amazon S3 bucket with object lock actions.
    /// </summary>
    /// <param name="bucketName">The name of the bucket to create.</param>
    /// <param name="enableObjectLock">True to enable object lock on the bucket.</
param>
    /// <returns>True if successful.</returns>
    public async Task<bool> CreateBucketWithObjectLock(string bucketName, bool
 enableObjectLock)
    {
        Console.WriteLine($"\tCreating bucket {bucketName} with object lock
 {enableObjectLock}.");
        try
        {
            var request = new PutBucketRequest
            {
                BucketName = bucketName,
                UseClientRegion = true,
                ObjectLockEnabledForBucket = enableObjectLock,
            };

            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、「 API リファレンスCreateBucket」の「」を参照してください。 AWS SDK for .NET

`DeleteBucket`

次の例は、DeleteBucket を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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="client">An initialized Amazon S3 client object.</param>
        /// <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 static async Task<bool> DeleteBucketAsync(IAmazonS3 client, string
 bucketName)
        {
            var request = new DeleteBucketRequest
            {
                BucketName = bucketName,
            };

            var response = await client.DeleteBucketAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- API の詳細については、「 API リファレンスDeleteBucket」の「」を参照してください。 AWS SDK for .NET

DeleteBucketCors

次の例は、DeleteBucketCors を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        /// <summary>
        /// Deletes a CORS configuration from an Amazon S3 bucket.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used
        /// to delete the CORS configuration from the bucket.</param>
        private static async Task DeleteCORSConfigurationAsync(AmazonS3Client
client)
        {
            DeleteCORSConfigurationRequest request = new
DeleteCORSConfigurationRequest()
            {
                BucketName = BucketName,
            };
            await client.DeleteCORSConfigurationAsync(request);
        }
```

- API の詳細については、「 API リファレンスDeleteBucketCors」の「」を参照してください。AWS SDK for .NET

DeleteBucketLifecycle

次の例は、DeleteBucketLifecycle を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        /// <summary>
        /// This method removes the Lifecycle configuration from the named
        /// S3 bucket.
        /// </summary>
        /// <param name="client">The S3 client object used to call
        /// the RemoveLifecycleConfigAsync method.</param>
        /// <param name="bucketName">A string representing the name of the
        /// S3 bucket from which the configuration will be removed.</param>
        public static async Task RemoveLifecycleConfigAsync(IAmazonS3 client, string
bucketName)
        {
            var request = new DeleteLifecycleConfigurationRequest()
            {
                BucketName = bucketName,
            };
            await client.DeleteLifecycleConfigurationAsync(request);
        }
```

- API の詳細については、「 API リファレンスDeleteBucketLifecycle」の「」を参照してください。 AWS SDK for .NET

`DeleteObject`

次の例は、`DeleteObject` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

バージョニング非対応の S3 バケットからオブジェクトを削除します。

```
using System;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example shows how to delete an object from a non-versioned Amazon
/// Simple Storage Service (Amazon S3) bucket.
/// </summary>
public class DeleteObject
{
    /// <summary>
    /// The Main method initializes the necessary variables and then calls
    /// the DeleteObjectNonVersionedBucketAsync method to delete the object
    /// named by the keyName parameter.
    /// </summary>
    public static async Task Main()
    {
        const string bucketName = "doc-example-bucket";
        const string keyName = "testfile.txt";

        // If the Amazon S3 bucket is located in an AWS Region other than the
        // Region of the default account, define the AWS Region for the
        // Amazon S3 bucket in your call to the AmazonS3Client constructor.
        // For example RegionEndpoint.USWest2.
        IAmazonS3 client = new AmazonS3Client();
        await DeleteObjectNonVersionedBucketAsync(client, bucketName, keyName);
    }

    /// <summary>
    /// The DeleteObjectNonVersionedBucketAsync takes care of deleting the
    /// desired object from the named bucket.
    /// </summary>
```

```
        /// <param name="client">An initialized Amazon S3 client used to delete
        /// an object from an Amazon S3 bucket.</param>
        /// <param name="bucketName">The name of the bucket from which the
        /// object will be deleted.</param>
        /// <param name="keyName">The name of the object to delete.</param>
        public static async Task DeleteObjectNonVersionedBucketAsync(IAmazonS3
client, string bucketName, string keyName)
        {
            try
            {
                var deleteObjectRequest = new DeleteObjectRequest
                {
                    BucketName = bucketName,
                    Key = keyName,
                };

                Console.WriteLine($"Deleting object: {keyName}");
                await client.DeleteObjectAsync(deleteObjectRequest);
                Console.WriteLine($"Object: {keyName} deleted from {bucketName}.");
            }
            catch (AmazonS3Exception ex)
            {
                Console.WriteLine($"Error encountered on server.
Message:'{ex.Message}' when deleting an object.");
            }
        }
    }
```

バージョニングされた S3 バケットからオブジェクトを削除します。

```
    using System;
    using System.Threading.Tasks;
    using Amazon.S3;
    using Amazon.S3.Model;

    /// <summary>
    /// This example creates an object in an Amazon Simple Storage Service
    /// (Amazon S3) bucket and then deletes the object version that was
    /// created.
    /// </summary>
    public class DeleteObjectVersion
```

```
    {
        public static async Task Main()
        {
            string bucketName = "doc-example-bucket";
            string keyName = "verstioned-object.txt";

            // If the AWS Region of the default user is different from the AWS
            // Region of the Amazon S3 bucket, pass the AWS Region of the
            // bucket region to the Amazon S3 client object's constructor.
            // Define it like this:
            //      RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
            IAmazonS3 client = new AmazonS3Client();

            await CreateAndDeleteObjectVersionAsync(client, bucketName, keyName);
        }

        /// <summary>
        /// This method creates and then deletes a versioned object.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used to
        /// create and delete the object.</param>
        /// <param name="bucketName">The name of the Amazon S3 bucket where the
        /// object will be created and deleted.</param>
        /// <param name="keyName">The key name of the object to create.</param>
        public static async Task CreateAndDeleteObjectVersionAsync(IAmazonS3 client,
string bucketName, string keyName)
        {
            try
            {
                // Add a sample object.
                string versionID = await PutAnObject(client, bucketName, keyName);

                // Delete the object by specifying an object key and a version ID.
                DeleteObjectRequest request = new DeleteObjectRequest()
                {
                    BucketName = bucketName,
                    Key = keyName,
                    VersionId = versionID,
                };

                Console.WriteLine("Deleting an object");
                await client.DeleteObjectAsync(request);
            }
            catch (AmazonS3Exception ex)
```

```
            {
                Console.WriteLine($"Error: {ex.Message}");
            }
        }

        /// <summary>
        /// This method is used to create the temporary Amazon S3 object.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 object which will be used
        /// to create the temporary Amazon S3 object.</param>
        /// <param name="bucketName">The name of the Amazon S3 bucket where the
 object
        /// will be created.</param>
        /// <param name="objectKey">The name of the Amazon S3 object co create.</
param>
        /// <returns>The Version ID of the created object.</returns>
        public static async Task<string> PutAnObject(IAmazonS3 client, string
 bucketName, string objectKey)
        {
            PutObjectRequest request = new PutObjectRequest()
            {
                BucketName = bucketName,
                Key = objectKey,
                ContentBody = "This is the content body!",
            };

            PutObjectResponse response = await client.PutObjectAsync(request);
            return response.VersionId;
        }
    }
```

- API の詳細については、「 API リファレンスDeleteObject」の「」を参照してください。 AWS SDK for .NET

DeleteObjects

次の例は、DeleteObjects を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

S3 バケットからすべてのオブジェクトを削除します。

```
        /// <summary>
        /// Delete all of the objects stored in an existing Amazon S3 bucket.
        /// </summary>
        /// <param name="client">An initialized Amazon S3 client object.</param>
        /// <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 static async Task<bool> DeleteBucketContentsAsync(IAmazonS3 client,
string bucketName)
        {
            // Iterate over the contents of the bucket and delete all objects.
            var request = new ListObjectsV2Request
            {
                BucketName = bucketName,
            };

            try
            {
                ListObjectsV2Response response;

                do
                {
                    response = await client.ListObjectsV2Async(request);
                    response.S3Objects
                        .ForEach(async obj => await
client.DeleteObjectAsync(bucketName, obj.Key));

                    // If the response is truncated, set the request
ContinuationToken
                    // from the NextContinuationToken property of the response.
                    request.ContinuationToken = response.NextContinuationToken;
```

```
            }
            while (response.IsTruncated);

            return true;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error deleting objects: {ex.Message}");
            return false;
        }
    }
```

バージョニング非対応の S3 バケットから複数のオブジェクトを削除します。

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example shows how to delete multiple objects from an Amazon Simple
/// Storage Service (Amazon S3) bucket.
/// </summary>
public class DeleteMultipleObjects
{
    /// <summary>
    /// The Main method initializes the Amazon S3 client and the name of
    /// the bucket and then passes those values to MultiObjectDeleteAsync.
    /// </summary>
    public static async Task Main()
    {
        const string bucketName = "doc-example-bucket";

        // If the Amazon S3 bucket from which you wish to delete objects is not
        // located in the same AWS Region as the default user, define the
        // AWS Region for the Amazon S3 bucket as a parameter to the client
        // constructor.
        IAmazonS3 s3Client = new AmazonS3Client();

        await MultiObjectDeleteAsync(s3Client, bucketName);
```

```
        }

        /// <summary>
        /// This method uses the passed Amazon S3 client to first create and then
        /// delete three files from the named bucket.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used to
call
        /// Amazon S3 methods.</param>
        /// <param name="bucketName">The name of the Amazon S3 bucket where objects
        /// will be created and then deleted.</param>
        public static async Task MultiObjectDeleteAsync(IAmazonS3 client, string
bucketName)
        {
            // Create three sample objects which we will then delete.
            var keysAndVersions = await PutObjectsAsync(client, 3, bucketName);

            // Now perform the multi-object delete, passing the key names and
            // version IDs. Since we are working with a non-versioned bucket,
            // the object keys collection includes null version IDs.
            DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest
            {
                BucketName = bucketName,
                Objects = keysAndVersions,
            };

            // You can add a specific object key to the delete request using the
            // AddKey method of the multiObjectDeleteRequest.
            try
            {
                DeleteObjectsResponse response = await
client.DeleteObjectsAsync(multiObjectDeleteRequest);
                Console.WriteLine("Successfully deleted all the {0} items",
response.DeletedObjects.Count);
            }
            catch (DeleteObjectsException e)
            {
                PrintDeletionErrorStatus(e);
            }
        }

        /// <summary>
        /// Prints the list of errors raised by the call to DeleteObjectsAsync.
        /// </summary>
```

```
        /// <param name="ex">A collection of exceptions returned by the call to
        /// DeleteObjectsAsync.</param>
        public static void PrintDeletionErrorStatus(DeleteObjectsException ex)
        {
            DeleteObjectsResponse errorResponse = ex.Response;
            Console.WriteLine("x {0}", errorResponse.DeletedObjects.Count);

            Console.WriteLine($"Successfully deleted
 {errorResponse.DeletedObjects.Count}.");
            Console.WriteLine($"No. of objects failed to delete =
 {errorResponse.DeleteErrors.Count}");

            Console.WriteLine("Printing error data...");
            foreach (DeleteError deleteError in errorResponse.DeleteErrors)
            {
                Console.WriteLine($"Object Key:
 {deleteError.Key}\t{deleteError.Code}\t{deleteError.Message}");
            }
        }

        /// <summary>
        /// This method creates simple text file objects that can be used in
        /// the delete method.
        /// </summary>
        /// <param name="client">The Amazon S3 client used to call PutObjectAsync.</
param>
        /// <param name="number">The number of objects to create.</param>
        /// <param name="bucketName">The name of the bucket where the objects
        /// will be created.</param>
        /// <returns>A list of keys (object keys) and versions that the calling
        /// method will use to delete the newly created files.</returns>
        public static async Task<List<KeyVersion>> PutObjectsAsync(IAmazonS3 client,
 int number, string bucketName)
        {
            List<KeyVersion> keys = new List<KeyVersion>();
            for (int i = 0; i < number; i++)
            {
                string key = "ExampleObject-" + new System.Random().Next();
                PutObjectRequest request = new PutObjectRequest
                {
                    BucketName = bucketName,
                    Key = key,
                    ContentBody = "This is the content body!",
                };
```

```
            PutObjectResponse response = await client.PutObjectAsync(request);

            // For non-versioned bucket operations, we only need the
            // object key.
            KeyVersion keyVersion = new KeyVersion
            {
                Key = key,
            };
            keys.Add(keyVersion);
        }

        return keys;
    }
}
```

バージョニングされた S3 バケットから複数のオブジェクトを削除します。

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example shows how to delete objects in a version-enabled Amazon
/// Simple StorageService (Amazon S3) bucket.
/// </summary>
public class DeleteMultipleObjects
{
    public static async Task Main()
    {
        string bucketName = "doc-example-bucket";

        // If the AWS Region for your Amazon S3 bucket is different from
        // the AWS Region of the default user, define the AWS Region for
        // the Amazon S3 bucket and pass it to the client constructor
        // like this:
        // RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        IAmazonS3 s3Client;
```

```
            s3Client = new AmazonS3Client();
            await DeleteMultipleObjectsFromVersionedBucketAsync(s3Client,
bucketName);
        }

        /// <summary>
        /// This method removes multiple versions and objects from a
        /// version-enabled Amazon S3 bucket.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used to
call
        /// DeleteObjectVersionsAsync, DeleteObjectsAsync, and
        /// RemoveDeleteMarkersAsync.</param>
        /// <param name="bucketName">The name of the bucket from which to delete
        /// objects.</param>
        public static async Task
DeleteMultipleObjectsFromVersionedBucketAsync(IAmazonS3 client, string bucketName)
        {
            // Delete objects (specifying object version in the request).
            await DeleteObjectVersionsAsync(client, bucketName);

            // Delete objects (without specifying object version in the request).
            var deletedObjects = await DeleteObjectsAsync(client, bucketName);

            // Additional exercise - remove the delete markers Amazon S3 returned
from
            // the preceding response. This results in the objects reappearing
            // in the bucket (you can verify the appearance/disappearance of
            // objects in the console).
            await RemoveDeleteMarkersAsync(client, bucketName, deletedObjects);
        }

        /// <summary>
        /// Creates and then deletes non-versioned Amazon S3 objects and then
deletes
        /// them again. The method returns a list of the Amazon S3 objects deleted.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used to
call
        /// PubObjectsAsync and NonVersionedDeleteAsync.</param>
        /// <param name="bucketName">The name of the bucket where the objects
        /// will be created and then deleted.</param>
        /// <returns>A list of DeletedObjects.</returns>
```

```
        public static async Task<List<DeletedObject>> DeleteObjectsAsync(IAmazonS3
client, string bucketName)
        {
            // Upload the sample objects.
            var keysAndVersions2 = await PutObjectsAsync(client, bucketName, 3);

            // Delete objects using only keys. Amazon S3 creates a delete marker and
            // returns its version ID in the response.
            List<DeletedObject> deletedObjects = await
NonVersionedDeleteAsync(client, bucketName, keysAndVersions2);
            return deletedObjects;
        }

        /// <summary>
        /// This method creates several temporary objects and then deletes them.
        /// </summary>
        /// <param name="client">The S3 client.</param>
        /// <param name="bucketName">Name of the bucket.</param>
        /// <returns>Async task.</returns>
        public static async Task DeleteObjectVersionsAsync(IAmazonS3 client, string
bucketName)
        {
            // Upload the sample objects.
            var keysAndVersions1 = await PutObjectsAsync(client, bucketName, 3);

            // Delete the specific object versions.
            await VersionedDeleteAsync(client, bucketName, keysAndVersions1);
        }

        /// <summary>
        /// Displays the list of information about deleted files to the console.
        /// </summary>
        /// <param name="e">Error information from the delete process.</param>
        private static void DisplayDeletionErrors(DeleteObjectsException e)
        {
            var errorResponse = e.Response;
            Console.WriteLine($"No. of objects successfully deleted =
{errorResponse.DeletedObjects.Count}");
            Console.WriteLine($"No. of objects failed to delete =
{errorResponse.DeleteErrors.Count}");
            Console.WriteLine("Printing error data...");
            foreach (var deleteError in errorResponse.DeleteErrors)
            {
```

```
                Console.WriteLine($"Object Key:
 {deleteError.Key}\t{deleteError.Code}\t{deleteError.Message}");
            }
        }

        /// <summary>
        /// Delete multiple objects from a version-enabled bucket.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used to
 call
        /// DeleteObjectVersionsAsync, DeleteObjectsAsync, and
        /// RemoveDeleteMarkersAsync.</param>
        /// <param name="bucketName">The name of the bucket from which to delete
        /// objects.</param>
        /// <param name="keys">A list of key names for the objects to delete.</
param>
        private static async Task VersionedDeleteAsync(IAmazonS3 client, string
 bucketName, List<KeyVersion> keys)
        {
            var multiObjectDeleteRequest = new DeleteObjectsRequest
            {
                BucketName = bucketName,
                Objects = keys, // This includes the object keys and specific
 version IDs.
            };

            try
            {
                Console.WriteLine("Executing VersionedDelete...");
                DeleteObjectsResponse response = await
 client.DeleteObjectsAsync(multiObjectDeleteRequest);
                Console.WriteLine($"Successfully deleted all the
 {response.DeletedObjects.Count} items");
            }
            catch (DeleteObjectsException ex)
            {
                DisplayDeletionErrors(ex);
            }
        }

        /// <summary>
        /// Deletes multiple objects from a non-versioned Amazon S3 bucket.
        /// </summary>
```

```
        /// <param name="client">The initialized Amazon S3 client object used to
 call
        /// DeleteObjectVersionsAsync, DeleteObjectsAsync, and
        /// RemoveDeleteMarkersAsync.</param>
        /// <param name="bucketName">The name of the bucket from which to delete
        /// objects.</param>
        /// <param name="keys">A list of key names for the objects to delete.</
param>
        /// <returns>A list of the deleted objects.</returns>
        private static async Task<List<DeletedObject>>
 NonVersionedDeleteAsync(IAmazonS3 client, string bucketName, List<KeyVersion> keys)
        {
            // Create a request that includes only the object key names.
            DeleteObjectsRequest multiObjectDeleteRequest = new
 DeleteObjectsRequest();
            multiObjectDeleteRequest.BucketName = bucketName;

            foreach (var key in keys)
            {
                multiObjectDeleteRequest.AddKey(key.Key);
            }

            // Execute DeleteObjectsAsync.
            // The DeleteObjectsAsync method adds a delete marker for each
            // object deleted. You can verify that the objects were removed
            // using the Amazon S3 console.
            DeleteObjectsResponse response;
            try
            {
                Console.WriteLine("Executing NonVersionedDelete...");
                response = await
 client.DeleteObjectsAsync(multiObjectDeleteRequest);
                Console.WriteLine("Successfully deleted all the {0} items",
 response.DeletedObjects.Count);
            }
            catch (DeleteObjectsException ex)
            {
                DisplayDeletionErrors(ex);
                throw; // Some deletions failed. Investigate before continuing.
            }

            // This response contains the DeletedObjects list which we use to delete
 the delete markers.
            return response.DeletedObjects;
```

```
        }

        /// <summary>
        /// Deletes the markers left after deleting the temporary objects.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used to
 call
        /// DeleteObjectVersionsAsync, DeleteObjectsAsync, and
        /// RemoveDeleteMarkersAsync.</param>
        /// <param name="bucketName">The name of the bucket from which to delete
        /// objects.</param>
        /// <param name="deletedObjects">A list of the objects that were deleted.</
param>
        private static async Task RemoveDeleteMarkersAsync(IAmazonS3 client, string
 bucketName, List<DeletedObject> deletedObjects)
        {
            var keyVersionList = new List<KeyVersion>();

            foreach (var deletedObject in deletedObjects)
            {
                KeyVersion keyVersion = new KeyVersion
                {
                    Key = deletedObject.Key,
                    VersionId = deletedObject.DeleteMarkerVersionId,
                };
                keyVersionList.Add(keyVersion);
            }

            // Create another request to delete the delete markers.
            var multiObjectDeleteRequest = new DeleteObjectsRequest
            {
                BucketName = bucketName,
                Objects = keyVersionList,
            };

            // Now, delete the delete marker to bring your objects back to the
 bucket.
            try
            {
                Console.WriteLine("Removing the delete markers .....");
                var deleteObjectResponse = await
 client.DeleteObjectsAsync(multiObjectDeleteRequest);
                Console.WriteLine($"Successfully deleted the
 {deleteObjectResponse.DeletedObjects.Count} delete markers");
```

```
            }
            catch (DeleteObjectsException ex)
            {
                DisplayDeletionErrors(ex);
            }
        }

        /// <summary>
        /// Create temporary Amazon S3 objects to show how object deletion wors in
an
        /// Amazon S3 bucket with versioning enabled.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used to
call
        /// PutObjectAsync to create temporary objects for the example.</param>
        /// <param name="bucketName">A string representing the name of the S3
        /// bucket where we will create the temporary objects.</param>
        /// <param name="number">The number of temporary objects to create.</param>
        /// <returns>A list of the KeyVersion objects.</returns>
        private static async Task<List<KeyVersion>> PutObjectsAsync(IAmazonS3
client, string bucketName, int number)
        {
            var keys = new List<KeyVersion>();

            for (var i = 0; i < number; i++)
            {
                string key = "ObjectToDelete-" + new System.Random().Next();
                PutObjectRequest request = new PutObjectRequest
                {
                    BucketName = bucketName,
                    Key = key,
                    ContentBody = "This is the content body!",
                };

                var response = await client.PutObjectAsync(request);
                KeyVersion keyVersion = new KeyVersion
                {
                    Key = key,
                    VersionId = response.VersionId,
                };

                keys.Add(keyVersion);
            }
```

```
            return keys;
        }
    }
```

- API の詳細については、「 API リファレンスDeleteObjects」の「」を参照してください。 AWS SDK for .NET

`GetBucketAcl`

次の例は、GetBucketAcl を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        /// <summary>
        /// Get the access control list (ACL) for the new bucket.
        /// </summary>
        /// <param name="client">The initialized client object used to get the
        /// access control list (ACL) of the bucket.</param>
        /// <param name="newBucketName">The name of the newly created bucket.</
param>
        /// <returns>An S3AccessControlList.</returns>
        public static async Task<S3AccessControlList> GetACLForBucketAsync(IAmazonS3
 client, string newBucketName)
        {
            // Retrieve bucket ACL to show that the ACL was properly applied to
            // the new bucket.
            GetACLResponse getACLResponse = await client.GetACLAsync(new
 GetACLRequest
            {
                BucketName = newBucketName,
            });
```

```
            return getACLResponse.AccessControlList;
        }
```

- API の詳細については、「 API リファレンスGetBucketAcl」の「」を参照してください。 AWS SDK for .NET

`GetBucketCors`

次の例は、GetBucketCors を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        /// <summary>
        /// Retrieve the CORS configuration applied to the Amazon S3 bucket.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used
        /// to retrieve the CORS configuration.</param>
        /// <returns>The created CORS configuration object.</returns>
        private static async Task<CORSConfiguration>
RetrieveCORSConfigurationAsync(AmazonS3Client client)
        {
            GetCORSConfigurationRequest request = new GetCORSConfigurationRequest()
            {
                BucketName = BucketName,
            };
            var response = await client.GetCORSConfigurationAsync(request);
            var configuration = response.Configuration;
            PrintCORSRules(configuration);
            return configuration;
        }
```

- API の詳細については、「 API リファレンスGetBucketCors」の「」を参照してください。 AWS SDK for .NET

`GetBucketLifecycleConfiguration`

次の例は、`GetBucketLifecycleConfiguration` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        /// <summary>
        /// Returns a configuration object for the supplied bucket name.
        /// </summary>
        /// <param name="client">The S3 client object used to call
        /// the GetLifecycleConfigurationAsync method.</param>
        /// <param name="bucketName">The name of the S3 bucket for which a
        /// configuration will be created.</param>
        /// <returns>Returns a new LifecycleConfiguration object.</returns>
        public static async Task<LifecycleConfiguration>
RetrieveLifecycleConfigAsync(IAmazonS3 client, string bucketName)
        {
            var request = new GetLifecycleConfigurationRequest()
            {
                BucketName = bucketName,
            };
            var response = await client.GetLifecycleConfigurationAsync(request);
            var configuration = response.Configuration;
            return configuration;
        }
```

- API の詳細については、「 API リファレンスGetBucketLifecycleConfiguration」の「」を参照してください。 AWS SDK for .NET

GetBucketWebsite

次の例は、GetBucketWebsite を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
                // Get the website configuration.
                GetBucketWebsiteRequest getRequest = new GetBucketWebsiteRequest()
                {
                    BucketName = bucketName,
                };
                GetBucketWebsiteResponse getResponse = await
client.GetBucketWebsiteAsync(getRequest);
                Console.WriteLine($"Index document:
{getResponse.WebsiteConfiguration.IndexDocumentSuffix}");
                Console.WriteLine($"Error document:
{getResponse.WebsiteConfiguration.ErrorDocument}");
```

- API の詳細については、「 API リファレンスGetBucketWebsite」の「」を参照してください。 AWS SDK for .NET

GetObject

次の例は、GetObject を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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="client">An initialized Amazon S3 client object.</param>
        /// <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 static async Task<bool> DownloadObjectFromBucketAsync(
            IAmazonS3 client,
            string bucketName,
            string objectName,
            string filePath)
        {
            // Create a GetObject request
            var request = new GetObjectRequest
            {
                BucketName = bucketName,
                Key = objectName,
            };

            // Issue request and remember to dispose of the response
            using GetObjectResponse response = await client.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、「 API リファレンスGetObject」の「」を参照してください。 AWS SDK for .NET

`GetObjectLegalHold`

次の例は、`GetObjectLegalHold` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get the legal hold details for an S3 object.
    /// </summary>
    /// <param name="bucketName">The bucket of the object.</param>
    /// <param name="objectKey">The object key.</param>
    /// <returns>The object legal hold details.</returns>
    public async Task<ObjectLockLegalHold> GetObjectLegalHold(string bucketName,
        string objectKey)
    {
        try
        {
            var request = new GetObjectLegalHoldRequest()
            {
                BucketName = bucketName,
                Key = objectKey
            };

            var response = await _amazonS3.GetObjectLegalHoldAsync(request);
            Console.WriteLine($"\tObject legal hold for {objectKey} in {bucketName}:
" +
                              $"\n\tStatus: {response.LegalHold.Status}");
            return response.LegalHold;
        }
        catch (AmazonS3Exception ex)
```

```
        {
            Console.WriteLine($"\tUnable to fetch legal hold: '{ex.Message}'");
            return new ObjectLockLegalHold();
        }
    }
```

- API の詳細については、「 API リファレンスGetObjectLegalHold」の「」を参照してください。 AWS SDK for .NET

`GetObjectLockConfiguration`

次の例は、`GetObjectLockConfiguration` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get the object lock configuration details for an S3 bucket.
    /// </summary>
    /// <param name="bucketName">The bucket to get details.</param>
    /// <returns>The bucket's object lock configuration details.</returns>
    public async Task<ObjectLockConfiguration>
GetBucketObjectLockConfiguration(string bucketName)
    {
        try
        {
            var request = new GetObjectLockConfigurationRequest()
            {
                BucketName = bucketName
            };

            var response = await _amazonS3.GetObjectLockConfigurationAsync(request);
            Console.WriteLine($"\tBucket object lock config for {bucketName} in
{bucketName}: " +
```

```
                                $"\n\tEnabled:
{response.ObjectLockConfiguration.ObjectLockEnabled}" +
                                $"\n\tRule:
{response.ObjectLockConfiguration.Rule?.DefaultRetention}");

            return response.ObjectLockConfiguration;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"\tUnable to fetch object lock config:
'{ex.Message}'");
            return new ObjectLockConfiguration();
        }
    }
```

- API の詳細については、「 API リファレンスGetObjectLockConfiguration」の「」を参照してください。 AWS SDK for .NET

`GetObjectRetention`

次の例は、`GetObjectRetention` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get the retention period for an S3 object.
    /// </summary>
    /// <param name="bucketName">The bucket of the object.</param>
    /// <param name="objectKey">The object key.</param>
    /// <returns>The object retention details.</returns>
    public async Task<ObjectLockRetention> GetObjectRetention(string bucketName,
        string objectKey)
    {
        try
```

```
        {
            var request = new GetObjectRetentionRequest()
            {
                BucketName = bucketName,
                Key = objectKey
            };

            var response = await _amazonS3.GetObjectRetentionAsync(request);
            Console.WriteLine($"\tObject retention for {objectKey} in {bucketName}:
" +
                              $"\n\t{response.Retention.Mode} until
{response.Retention.RetainUntilDate:d}.");
            return response.Retention;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"\tUnable to fetch object lock retention:
'{ex.Message}'");
            return new ObjectLockRetention();
        }
    }
```

- API の詳細については、「 API リファレンスGetObjectRetention」の「」を参照してください。 AWS SDK for .NET

`ListBuckets`

次の例は、`ListBuckets` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
namespace ListBucketsExample
{
    using System;
```

```
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon.S3;
    using Amazon.S3.Model;

    /// <summary>
    /// This example uses the AWS SDK for .NET to list the Amazon Simple Storage
    /// Service (Amazon S3) buckets belonging to the default account.
    /// </summary>
    public class ListBuckets
    {
        private static IAmazonS3 _s3Client;

        /// <summary>
        /// Get a list of the buckets owned by the default user.
        /// </summary>
        /// <param name="client">An initialized Amazon S3 client object.</param>
        /// <returns>The response from the ListingBuckets call that contains a
        /// list of the buckets owned by the default user.</returns>
        public static async Task<ListBucketsResponse> GetBuckets(IAmazonS3 client)
        {
            return await client.ListBucketsAsync();
        }

        /// <summary>
        /// This method lists the name and creation date for the buckets in
        /// the passed List of S3 buckets.
        /// </summary>
        /// <param name="bucketList">A List of S3 bucket objects.</param>
        public static void DisplayBucketList(List<S3Bucket> bucketList)
        {
            bucketList
                .ForEach(b => Console.WriteLine($"Bucket name: {b.BucketName},
created on: {b.CreationDate}"));
        }

        public static async Task Main()
        {
            // The client uses the AWS Region of the default user.
            // If the Region where the buckets were created is different,
            // pass the Region to the client constructor. For example:
            // _s3Client = new AmazonS3Client(RegionEndpoint.USEast1);
            _s3Client = new AmazonS3Client();
            var response = await GetBuckets(_s3Client);
```

```
            DisplayBucketList(response.Buckets);
        }
    }
}
```

- API の詳細については、「 API リファレンスListBuckets」の「」を参照してください。 AWS SDK for .NET

`ListObjectVersions`

次の例は、`ListObjectVersions` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    using System;
    using System.Threading.Tasks;
    using Amazon.S3;
    using Amazon.S3.Model;

    /// <summary>
    /// This example lists the versions of the objects in a version enabled
    /// Amazon Simple Storage Service (Amazon S3) bucket.
    /// </summary>
    public class ListObjectVersions
    {
        public static async Task Main()
        {
            string bucketName = "doc-example-bucket";

            // If the AWS Region where your bucket is defined is different from
            // the AWS Region where the Amazon S3 bucket is defined, pass the
constant
            // for the AWS Region to the client constructor like this:
```

```
            //      var client = new AmazonS3Client(RegionEndpoint.USWest2);
            IAmazonS3 client = new AmazonS3Client();
            await GetObjectListWithAllVersionsAsync(client, bucketName);
        }

        /// <summary>
        /// This method lists all versions of the objects within an Amazon S3
        /// version enabled bucket.
        /// </summary>
        /// <param name="client">The initialized client object used to call
        /// ListVersionsAsync.</param>
        /// <param name="bucketName">The name of the version enabled Amazon S3
 bucket
        /// for which you want to list the versions of the contained objects.</
param>
        public static async Task GetObjectListWithAllVersionsAsync(IAmazonS3 client,
 string bucketName)
        {
            try
            {
                // When you instantiate the ListVersionRequest, you can
                // optionally specify a key name prefix in the request
                // if you want a list of object versions of a specific object.

                // For this example we set a small limit in MaxKeys to return
                // a small list of versions.
                ListVersionsRequest request = new ListVersionsRequest()
                {
                    BucketName = bucketName,
                    MaxKeys = 2,
                };

                do
                {
                    ListVersionsResponse response = await
 client.ListVersionsAsync(request);

                    // Process response.
                    foreach (S3ObjectVersion entry in response.Versions)
                    {
                        Console.WriteLine($"key: {entry.Key} size: {entry.Size}");
                    }

                    // If response is truncated, set the marker to get the next
```

```
                    // set of keys.
                    if (response.IsTruncated)
                    {
                        request.KeyMarker = response.NextKeyMarker;
                        request.VersionIdMarker = response.NextVersionIdMarker;
                    }
                    else
                    {
                        request = null;
                    }
                }
                while (request != null);
            }
            catch (AmazonS3Exception ex)
            {
                Console.WriteLine($"Error: '{ex.Message}'");
            }
        }
    }
```

- API の詳細については、「 API リファレンスListObjectVersions」の「」を参照してください。AWS SDK for .NET

`ListObjectsV2`

次の例は、`ListObjectsV2` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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="client">An initialized Amazon S3 client object.</param>
        /// <param name="bucketName">The name of the bucket for which to list
        /// the contents.</param>
        /// <returns>A boolean value indicating the success or failure of the
        /// copy operation.</returns>
        public static async Task<bool> ListBucketContentsAsync(IAmazonS3 client,
 string bucketName)
        {
            try
            {
                var request = new ListObjectsV2Request
                {
                    BucketName = bucketName,
                    MaxKeys = 5,
                };

                Console.WriteLine("--------------------------------------");
                Console.WriteLine($"Listing the contents of {bucketName}:");
                Console.WriteLine("--------------------------------------");

                ListObjectsV2Response response;

                do
                {
                    response = await client.ListObjectsV2Async(request);

                    response.S3Objects
                        .ForEach(obj => Console.WriteLine($"{obj.Key,-35}
{obj.LastModified.ToShortDateString(),10}{obj.Size,10}"));

                    // If the response is truncated, set the request
 ContinuationToken
                    // from the NextContinuationToken property of the response.
                    request.ContinuationToken = response.NextContinuationToken;
                }
                while (response.IsTruncated);

                return true;
            }
            catch (AmazonS3Exception ex)
            {
                Console.WriteLine($"Error encountered on server.
 Message:'{ex.Message}' getting list of objects.");
                return false;
```

```
            }
        }
```

ページネーターを使用してオブジェクトを一覧表示します。

```
    using System;
    using System.Threading.Tasks;
    using Amazon.S3;
    using Amazon.S3.Model;

    /// <summary>
    /// The following example lists objects in an Amazon Simple Storage
    /// Service (Amazon S3) bucket.
    /// </summary>
    public class ListObjectsPaginator
    {
        private const string BucketName = "doc-example-bucket";

        public static async Task Main()
        {
            IAmazonS3 s3Client = new AmazonS3Client();

            Console.WriteLine($"Listing the objects contained in {BucketName}:\n");
            await ListingObjectsAsync(s3Client, BucketName);
        }

        /// <summary>
        /// This method uses a paginator to retrieve the list of objects in an
        /// an Amazon S3 bucket.
        /// </summary>
        /// <param name="client">An Amazon S3 client object.</param>
        /// <param name="bucketName">The name of the S3 bucket whose objects
        /// you want to list.</param>
        public static async Task ListingObjectsAsync(IAmazonS3 client, string
bucketName)
        {
            var listObjectsV2Paginator = client.Paginators.ListObjectsV2(new
ListObjectsV2Request
            {
                BucketName = bucketName,
            });
```

```
        await foreach (var response in listObjectsV2Paginator.Responses)
        {
            Console.WriteLine($"HttpStatusCode: {response.HttpStatusCode}");
            Console.WriteLine($"Number of Keys: {response.KeyCount}");
            foreach (var entry in response.S3Objects)
            {
                Console.WriteLine($"Key = {entry.Key} Size = {entry.Size}");
            }
        }
    }
}
```

- API の詳細については、「 API リファレンス」のListObjectsV2」を参照してください。 AWS SDK for .NET

`PutBucketAccelerateConfiguration`

次の例は、PutBucketAccelerateConfiguration を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
using System;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// Amazon Simple Storage Service (Amazon S3) Transfer Acceleration is a
/// bucket-level feature that enables you to perform faster data transfers
/// to Amazon S3. This example shows how to configure Transfer
/// Acceleration.
/// </summary>
```

```
    public class TransferAcceleration
    {
        /// <summary>
        /// The main method initializes the client object and sets the
        /// Amazon Simple Storage Service (Amazon S3) bucket name before
        /// calling EnableAccelerationAsync.
        /// </summary>
        public static async Task Main()
        {
            var s3Client = new AmazonS3Client();
            const string bucketName = "doc-example-bucket";

            await EnableAccelerationAsync(s3Client, bucketName);
        }

        /// <summary>
        /// This method sets the configuration to enable transfer acceleration
        /// for the bucket referred to in the bucketName parameter.
        /// </summary>
        /// <param name="client">An Amazon S3 client used to enable the
        /// acceleration on an Amazon S3 bucket.</param>
        /// <param name="bucketName">The name of the Amazon S3 bucket for which the
        /// method will be enabling acceleration.</param>
        private static async Task EnableAccelerationAsync(AmazonS3Client client,
string bucketName)
        {
            try
            {
                var putRequest = new PutBucketAccelerateConfigurationRequest
                {
                    BucketName = bucketName,
                    AccelerateConfiguration = new AccelerateConfiguration
                    {
                        Status = BucketAccelerateStatus.Enabled,
                    },
                };
                await client.PutBucketAccelerateConfigurationAsync(putRequest);

                var getRequest = new GetBucketAccelerateConfigurationRequest
                {
                    BucketName = bucketName,
                };
                var response = await
client.GetBucketAccelerateConfigurationAsync(getRequest);
```

```
                Console.WriteLine($"Acceleration state = '{response.Status}' ");
            }
            catch (AmazonS3Exception ex)
            {
                Console.WriteLine($"Error occurred. Message:'{ex.Message}' when
 setting transfer acceleration");
            }
        }
    }
```

- API の詳細については、「 API リファレンスPutBucketAccelerateConfiguration」の「」を参照してください。 AWS SDK for .NET

PutBucketAcl

次の例は、PutBucketAcl を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        /// <summary>
        /// Creates an Amazon S3 bucket with an ACL to control access to the
        /// bucket and the objects stored in it.
        /// </summary>
        /// <param name="client">The initialized client object used to create
        /// an Amazon S3 bucket, with an ACL applied to the bucket.
        /// </param>
        /// <param name="region">The AWS Region where the bucket will be created.</
param>
        /// <param name="newBucketName">The name of the bucket to create.</param>
        /// <returns>A boolean value indicating success or failure.</returns>
```

```
        public static async Task<bool> CreateBucketUseCannedACLAsync(IAmazonS3
client, S3Region region, string newBucketName)
        {
            try
            {
                // Create a new Amazon S3 bucket with Canned ACL.
                var putBucketRequest = new PutBucketRequest()
                {
                    BucketName = newBucketName,
                    BucketRegion = region,
                    CannedACL = S3CannedACL.LogDeliveryWrite,
                };

                PutBucketResponse putBucketResponse = await
client.PutBucketAsync(putBucketRequest);

                return putBucketResponse.HttpStatusCode ==
System.Net.HttpStatusCode.OK;
            }
            catch (AmazonS3Exception ex)
            {
                Console.WriteLine($"Amazon S3 error: {ex.Message}");
            }

            return false;
        }
```

- API の詳細については、「 API リファレンスPutBucketAcl」の「」を参照してください。AWS SDK for .NET

PutBucketCors

次の例は、PutBucketCors を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        /// <summary>
        /// Add CORS configuration to the Amazon S3 bucket.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used
        /// to apply the CORS configuration to an Amazon S3 bucket.</param>
        /// <param name="configuration">The CORS configuration to apply.</param>
        private static async Task PutCORSConfigurationAsync(AmazonS3Client client,
CORSConfiguration configuration)
        {
            PutCORSConfigurationRequest request = new PutCORSConfigurationRequest()
            {
                BucketName = BucketName,
                Configuration = configuration,
            };

            _ = await client.PutCORSConfigurationAsync(request);
        }
```

- API の詳細については、「 API リファレンスPutBucketCors」の「」を参照してください。AWS SDK for .NET

`PutBucketLifecycleConfiguration`

次の例は、`PutBucketLifecycleConfiguration` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        /// <summary>
        /// Adds lifecycle configuration information to the S3 bucket named in
        /// the bucketName parameter.
        /// </summary>
        /// <param name="client">The S3 client used to call the
        /// PutLifecycleConfigurationAsync method.</param>
        /// <param name="bucketName">A string representing the S3 bucket to
        /// which configuration information will be added.</param>
        /// <param name="configuration">A LifecycleConfiguration object that
        /// will be applied to the S3 bucket.</param>
        public static async Task AddExampleLifecycleConfigAsync(IAmazonS3 client,
string bucketName, LifecycleConfiguration configuration)
        {
            var request = new PutLifecycleConfigurationRequest()
            {
                BucketName = bucketName,
                Configuration = configuration,
            };
            var response = await client.PutLifecycleConfigurationAsync(request);
        }
```

- API の詳細については、「 API リファレンスPutBucketLifecycleConfiguration」の「」を参照してください。 AWS SDK for .NET

PutBucketLogging

次の例は、PutBucketLogging を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
using System;
using System.IO;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;
using Microsoft.Extensions.Configuration;

/// <summary>
/// This example shows how to enable logging on an Amazon Simple Storage
/// Service (Amazon S3) bucket. You need to have two Amazon S3 buckets for
/// this example. The first is the bucket for which you wish to enable
/// logging, and the second is the location where you want to store the
/// logs.
/// </summary>
public class ServerAccessLogging
{
    private static IConfiguration _configuration = null!;

    public static async Task Main()
    {
        LoadConfig();

        string bucketName = _configuration["BucketName"];
        string logBucketName = _configuration["LogBucketName"];
        string logObjectKeyPrefix = _configuration["LogObjectKeyPrefix"];
        string accountId = _configuration["AccountId"];

        // If the AWS Region defined for your default user is different
        // from the Region where your Amazon S3 bucket is located,
        // pass the Region name to the Amazon S3 client object's constructor.
        // For example: RegionEndpoint.USWest2 or RegionEndpoint.USEast2.
        IAmazonS3 client = new AmazonS3Client();

        try
```

```
            {
                // Update bucket policy for target bucket to allow delivery of logs
to it.
                await SetBucketPolicyToAllowLogDelivery(
                    client,
                    bucketName,
                    logBucketName,
                    logObjectKeyPrefix,
                    accountId);

                // Enable logging on the source bucket.
                await EnableLoggingAsync(
                    client,
                    bucketName,
                    logBucketName,
                    logObjectKeyPrefix);
            }
            catch (AmazonS3Exception e)
            {
                Console.WriteLine($"Error: {e.Message}");
            }
        }

        /// <summary>
        /// This method grants appropriate permissions for logging to the
        /// Amazon S3 bucket where the logs will be stored.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client which will be used
        /// to apply the bucket policy.</param>
        /// <param name="sourceBucketName">The name of the source bucket.</param>
        /// <param name="logBucketName">The name of the bucket where logging
        /// information will be stored.</param>
        /// <param name="logPrefix">The logging prefix where the logs should be
delivered.</param>
        /// <param name="accountId">The account id of the account where the source
bucket exists.</param>
        /// <returns>Async task.</returns>
        public static async Task SetBucketPolicyToAllowLogDelivery(
            IAmazonS3 client,
            string sourceBucketName,
            string logBucketName,
            string logPrefix,
            string accountId)
        {
```

```
            var resourceArn = @"""arn:aws:s3:::" + logBucketName + "/" + logPrefix +
@"*""";

            var newPolicy = @"{
                                ""Statement"":[{
                                ""Sid"": ""S3ServerAccessLogsPolicy"",
                                ""Effect"": ""Allow"",
                                ""Principal"": { ""Service"":
""logging.s3.amazonaws.com"" },
                                ""Action"": [""s3:PutObject""],
                                ""Resource"": [" + resourceArn + @"],
                                ""Condition"": {
                                ""ArnLike"": { ""aws:SourceArn"": ""arn:aws:s3:::" +
sourceBucketName + @""" },
                                ""StringEquals"": { ""aws:SourceAccount"": """ +
accountId + @""" }
                                        }
                                    }]
                                }";
            Console.WriteLine($"The policy to apply to bucket {logBucketName} to
enable logging:");
            Console.WriteLine(newPolicy);

            PutBucketPolicyRequest putRequest = new PutBucketPolicyRequest
            {
                BucketName = logBucketName,
                Policy = newPolicy,
            };
            await client.PutBucketPolicyAsync(putRequest);
            Console.WriteLine("Policy applied.");
        }

        /// <summary>
        /// This method enables logging for an Amazon S3 bucket. Logs will be stored
        /// in the bucket you selected for logging. Selected prefix
        /// will be prepended to each log object.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client which will be used
        /// to configure and apply logging to the selected Amazon S3 bucket.</param>
        /// <param name="bucketName">The name of the Amazon S3 bucket for which you
        /// wish to enable logging.</param>
        /// <param name="logBucketName">The name of the Amazon S3 bucket where
logging
        /// information will be stored.</param>
```

```
        /// <param name="logObjectKeyPrefix">The prefix to prepend to each
        /// object key.</param>
        /// <returns>Async task.</returns>
        public static async Task EnableLoggingAsync(
            IAmazonS3 client,
            string bucketName,
            string logBucketName,
            string logObjectKeyPrefix)
        {
            Console.WriteLine($"Enabling logging for bucket {bucketName}.");
            var loggingConfig = new S3BucketLoggingConfig
            {
                TargetBucketName = logBucketName,
                TargetPrefix = logObjectKeyPrefix,
            };

            var putBucketLoggingRequest = new PutBucketLoggingRequest
            {
                BucketName = bucketName,
                LoggingConfig = loggingConfig,
            };
            await client.PutBucketLoggingAsync(putBucketLoggingRequest);
            Console.WriteLine($"Logging enabled.");
        }

        /// <summary>
        /// Loads configuration from settings files.
        /// </summary>
        public static void LoadConfig()
        {
            _configuration = new ConfigurationBuilder()
                .SetBasePath(Directory.GetCurrentDirectory())
                .AddJsonFile("settings.json") // Load settings from .json file.
                .AddJsonFile("settings.local.json", true) // Optionally, load local
settings.
                .Build();
        }
    }
```

- API の詳細については、「 API リファレンスPutBucketLogging」の「」を参照してください。AWS SDK for .NET

PutBucketNotificationConfiguration

次の例は、PutBucketNotificationConfiguration を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon.S3;
    using Amazon.S3.Model;

    /// <summary>
    /// This example shows how to enable notifications for an Amazon Simple
    /// Storage Service (Amazon S3) bucket.
    /// </summary>
    public class EnableNotifications
    {
        public static async Task Main()
        {
            const string bucketName = "doc-example-bucket1";
            const string snsTopic = "arn:aws:sns:us-east-2:0123456789ab:bucket-
notify";
            const string sqsQueue = "arn:aws:sqs:us-
east-2:0123456789ab:Example_Queue";

            IAmazonS3 client = new AmazonS3Client(Amazon.RegionEndpoint.USEast2);
            await EnableNotificationAsync(client, bucketName, snsTopic, sqsQueue);
        }

        /// <summary>
        /// This method makes the call to the PutBucketNotificationAsync method.
        /// </summary>
        /// <param name="client">An initialized Amazon S3 client used to call
        /// the PutBucketNotificationAsync method.</param>
        /// <param name="bucketName">The name of the bucket for which
        /// notifications will be turned on.</param>
```

```
        /// <param name="snsTopic">The ARN for the Amazon Simple Notification
        /// Service (Amazon SNS) topic associated with the S3 bucket.</param>
        /// <param name="sqsQueue">The ARN of the Amazon Simple Queue Service
        /// (Amazon SQS) queue to which notifications will be pushed.</param>
        public static async Task EnableNotificationAsync(
            IAmazonS3 client,
            string bucketName,
            string snsTopic,
            string sqsQueue)
        {
            try
            {
                // The bucket for which we are setting up notifications.
                var request = new PutBucketNotificationRequest()
                {
                    BucketName = bucketName,
                };

                // Defines the topic to use when sending a notification.
                var topicConfig = new TopicConfiguration()
                {
                    Events = new List<EventType> { EventType.ObjectCreatedCopy },
                    Topic = snsTopic,
                };
                request.TopicConfigurations = new List<TopicConfiguration>
                {
                    topicConfig,
                };
                request.QueueConfigurations = new List<QueueConfiguration>
                {
                    new QueueConfiguration()
                    {
                        Events = new List<EventType> { EventType.ObjectCreatedPut },
                        Queue = sqsQueue,
                    },
                };

                // Now apply the notification settings to the bucket.
                PutBucketNotificationResponse response = await
client.PutBucketNotificationAsync(request);
            }
            catch (AmazonS3Exception ex)
            {
                Console.WriteLine($"Error: {ex.Message}");
```

```
            }
        }
    }
```

- API の詳細については、「 API リファレンスPutBucketNotificationConfiguration」の「」を参照してください。 AWS SDK for .NET

`PutBucketWebsite`

次の例は、PutBucketWebsite を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
                // Put the website configuration.
                PutBucketWebsiteRequest putRequest = new PutBucketWebsiteRequest()
                {
                    BucketName = bucketName,
                    WebsiteConfiguration = new WebsiteConfiguration()
                    {
                        IndexDocumentSuffix = indexDocumentSuffix,
                        ErrorDocument = errorDocument,
                    },
                };
                PutBucketWebsiteResponse response = await
client.PutBucketWebsiteAsync(putRequest);
```

- API の詳細については、「 API リファレンスPutBucketWebsite」の「」を参照してください。 AWS SDK for .NET

PutObject

次の例は、PutObject を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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="client">An initialized Amazon S3 client object.</param>
        /// <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 static async Task<bool> UploadFileAsync(
            IAmazonS3 client,
            string bucketName,
            string objectName,
            string filePath)
        {
            var request = new PutObjectRequest
            {
                BucketName = bucketName,
                Key = objectName,
                FilePath = filePath,
            };

            var response = await client.PutObjectAsync(request);
            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"Successfully uploaded {objectName} to
{bucketName}.");
```

```
                return true;
            }
            else
            {
                Console.WriteLine($"Could not upload {objectName} to
{bucketName}.");
                return false;
            }
        }
```

サーバー側の暗号化を使用してオブジェクトをアップロードします。

```
    using System;
    using System.Threading.Tasks;
    using Amazon.S3;
    using Amazon.S3.Model;

    /// <summary>
    /// This example shows how to upload an object to an Amazon Simple Storage
    /// Service (Amazon S3) bucket with server-side encryption enabled.
    /// </summary>
    public class ServerSideEncryption
    {
        public static async Task Main()
        {
            string bucketName = "doc-example-bucket";
            string keyName = "samplefile.txt";

            // If the AWS Region defined for your default user is different
            // from the Region where your Amazon S3 bucket is located,
            // pass the Region name to the Amazon S3 client object's constructor.
            // For example: RegionEndpoint.USWest2.
            IAmazonS3 client = new AmazonS3Client();

            await WritingAnObjectAsync(client, bucketName, keyName);
        }

        /// <summary>
        /// Upload a sample object include a setting for encryption.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used to
```

```
        /// to upload a file and apply server-side encryption.</param>
        /// <param name="bucketName">The name of the Amazon S3 bucket where the
        /// encrypted object will reside.</param>
        /// <param name="keyName">The name for the object that you want to
        /// create in the supplied bucket.</param>
        public static async Task WritingAnObjectAsync(IAmazonS3 client, string
bucketName, string keyName)
        {
            try
            {
                var putRequest = new PutObjectRequest
                {
                    BucketName = bucketName,
                    Key = keyName,
                    ContentBody = "sample text",
                    ServerSideEncryptionMethod = ServerSideEncryptionMethod.AES256,
                };

                var putResponse = await client.PutObjectAsync(putRequest);

                // Determine the encryption state of an object.
                GetObjectMetadataRequest metadataRequest = new
GetObjectMetadataRequest
                {
                    BucketName = bucketName,
                    Key = keyName,
                };
                GetObjectMetadataResponse response = await
client.GetObjectMetadataAsync(metadataRequest);
                ServerSideEncryptionMethod objectEncryption =
response.ServerSideEncryptionMethod;

                Console.WriteLine($"Encryption method used: {0}",
objectEncryption.ToString());
            }
            catch (AmazonS3Exception ex)
            {
                Console.WriteLine($"Error: '{ex.Message}' when writing an object");
            }
        }
    }
```

- API の詳細については、「 API リファレンスPutObject」の「」を参照してください。 AWS SDK for .NET

`PutObjectLegalHold`

次の例は、PutObjectLegalHold を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Set or modify a legal hold on an object in an S3 bucket.
    /// </summary>
    /// <param name="bucketName">The bucket of the object.</param>
    /// <param name="objectKey">The key of the object.</param>
    /// <param name="holdStatus">The On or Off status for the legal hold.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> ModifyObjectLegalHold(string bucketName,
        string objectKey, ObjectLockLegalHoldStatus holdStatus)
    {
        try
        {
            var request = new PutObjectLegalHoldRequest()
            {
                BucketName = bucketName,
                Key = objectKey,
                LegalHold = new ObjectLockLegalHold()
                {
                    Status = holdStatus
                }
            };

            var response = await _amazonS3.PutObjectLegalHoldAsync(request);
            Console.WriteLine($"\tModified legal hold for {objectKey} in
{bucketName}.");
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

```
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"\tError modifying legal hold: '{ex.Message}'");
            return false;
        }
    }
```

- API の詳細については、「 API リファレンスPutObjectLegalHold」の「」を参照してください。 AWS SDK for .NET

PutObjectLockConfiguration

次の例は、PutObjectLockConfiguration を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

バケットのオブジェクトロック設定を指定します。

```
    /// <summary>
    /// Enable object lock on an existing bucket.
    /// </summary>
    /// <param name="bucketName">The name of the bucket to modify.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> EnableObjectLockOnBucket(string bucketName)
    {
        try
        {
            // First, enable Versioning on the bucket.
            await _amazonS3.PutBucketVersioningAsync(new
PutBucketVersioningRequest()
            {
                BucketName = bucketName,
                VersioningConfig = new S3BucketVersioningConfig()
                {
                    EnableMfaDelete = false,
```

```
                    Status = VersionStatus.Enabled
                }
            });

            var request = new PutObjectLockConfigurationRequest()
            {
                BucketName = bucketName,
                ObjectLockConfiguration = new ObjectLockConfiguration()
                {
                    ObjectLockEnabled = new ObjectLockEnabled("Enabled"),
                },
            };

            var response = await _amazonS3.PutObjectLockConfigurationAsync(request);
            Console.WriteLine($"\tAdded an object lock policy to bucket
{bucketName}.");
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error modifying object lock: '{ex.Message}'");
            return false;
        }
    }
```

バケットのデフォルトの保存期間を設定します。

```
    /// <summary>
    /// Set or modify a retention period on an S3 bucket.
    /// </summary>
    /// <param name="bucketName">The bucket to modify.</param>
    /// <param name="retention">The retention mode.</param>
    /// <param name="retainUntilDate">The date for retention until.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> ModifyBucketDefaultRetention(string bucketName, bool
enableObjectLock, ObjectLockRetentionMode retention, DateTime retainUntilDate)
    {
        var enabledString = enableObjectLock ? "Enabled" : "Disabled";
        var timeDifference = retainUntilDate.Subtract(DateTime.Now);
        try
        {
            // First, enable Versioning on the bucket.
```

```
            await _amazonS3.PutBucketVersioningAsync(new
PutBucketVersioningRequest()
            {
                BucketName = bucketName,
                VersioningConfig = new S3BucketVersioningConfig()
                {
                    EnableMfaDelete = false,
                    Status = VersionStatus.Enabled
                }
            });

            var request = new PutObjectLockConfigurationRequest()
            {
                BucketName = bucketName,
                ObjectLockConfiguration = new ObjectLockConfiguration()
                {
                    ObjectLockEnabled = new ObjectLockEnabled(enabledString),
                    Rule = new ObjectLockRule()
                    {
                        DefaultRetention = new DefaultRetention()
                        {
                            Mode = retention,
                            Days = timeDifference.Days // Can be specified in days
or years but not both.
                        }
                    }
                }
            };

            var response = await _amazonS3.PutObjectLockConfigurationAsync(request);
            Console.WriteLine($"\tAdded a default retention to bucket
{bucketName}.");
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"\tError modifying object lock: '{ex.Message}'");
            return false;
        }
    }
```

- API の詳細については、「 API リファレンスPutObjectLockConfiguration」の「」を参照してください。 AWS SDK for .NET

PutObjectRetention

次の例は、PutObjectRetention を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Set or modify a retention period on an object in an S3 bucket.
    /// </summary>
    /// <param name="bucketName">The bucket of the object.</param>
    /// <param name="objectKey">The key of the object.</param>
    /// <param name="retention">The retention mode.</param>
    /// <param name="retainUntilDate">The date retention expires.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> ModifyObjectRetentionPeriod(string bucketName,
        string objectKey, ObjectLockRetentionMode retention, DateTime
retainUntilDate)
    {
        try
        {
            var request = new PutObjectRetentionRequest()
            {
                BucketName = bucketName,
                Key = objectKey,
                Retention = new ObjectLockRetention()
                {
                    Mode = retention,
                    RetainUntilDate = retainUntilDate
                }
            };

            var response = await _amazonS3.PutObjectRetentionAsync(request);
```

```
            Console.WriteLine($"\tSet retention for {objectKey} in {bucketName}
until {retainUntilDate:d}.");
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"\tError modifying retention period:
'{ex.Message}'");
            return false;
        }
    }
```

- API の詳細については、「 API リファレンスPutObjectRetention」の「」を参照してください。 AWS SDK for .NET

`RestoreObject`

次の例は、`RestoreObject` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
using System;
using System.Threading.Tasks;
using Amazon;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example shows how to restore an archived object in an Amazon
/// Simple Storage Service (Amazon S3) bucket.
/// </summary>
public class RestoreArchivedObject
{
    public static void Main()
```

```
        {
            string bucketName = "doc-example-bucket";
            string objectKey = "archived-object.txt";

            // Specify your bucket region (an example region is shown).
            RegionEndpoint bucketRegion = RegionEndpoint.USWest2;

            IAmazonS3 client = new AmazonS3Client(bucketRegion);
            RestoreObjectAsync(client, bucketName, objectKey).Wait();
        }

        /// <summary>
        /// This method restores an archived object from an Amazon S3 bucket.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used to
call
        /// RestoreObjectAsync.</param>
        /// <param name="bucketName">A string representing the name of the
        /// bucket where the object was located before it was archived.</param>
        /// <param name="objectKey">A string representing the name of the
        /// archived object to restore.</param>
        public static async Task RestoreObjectAsync(IAmazonS3 client, string
bucketName, string objectKey)
        {
            try
            {
                var restoreRequest = new RestoreObjectRequest
                {
                    BucketName = bucketName,
                    Key = objectKey,
                    Days = 2,
                };
                RestoreObjectResponse response = await
client.RestoreObjectAsync(restoreRequest);

                // Check the status of the restoration.
                await CheckRestorationStatusAsync(client, bucketName, objectKey);
            }
            catch (AmazonS3Exception amazonS3Exception)
            {
                Console.WriteLine($"Error: {amazonS3Exception.Message}");
            }
        }
```

```
        /// <summary>
        /// This method retrieves the status of the object's restoration.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used to
call
        /// GetObjectMetadataAsync.</param>
        /// <param name="bucketName">A string representing the name of the Amazon
        /// S3 bucket which contains the archived object.</param>
        /// <param name="objectKey">A string representing the name of the
        /// archived object you want to restore.</param>
        public static async Task CheckRestorationStatusAsync(IAmazonS3 client,
string bucketName, string objectKey)
        {
            GetObjectMetadataRequest metadataRequest = new
GetObjectMetadataRequest()
            {
                BucketName = bucketName,
                Key = objectKey,
            };

            GetObjectMetadataResponse response = await
client.GetObjectMetadataAsync(metadataRequest);

            var restStatus = response.RestoreInProgress ? "in-progress" : "finished
or failed";
            Console.WriteLine($"Restoration status: {restStatus}");
        }
    }
```

- API の詳細については、「 API リファレンスRestoreObject」の「」を参照してください。AWS SDK for .NET

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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon S3 アクションを期間限定で実行できる署名付き URL を生成します。

```
    using System;
    using Amazon;
    using Amazon.S3;
    using Amazon.S3.Model;

    public class GenPresignedUrl
    {
        public static void Main()
        {
            const string bucketName = "doc-example-bucket";
            const string objectKey = "sample.txt";

            // Specify how long the presigned URL lasts, in hours
            const double timeoutDuration = 12;

            // Specify the AWS Region of your Amazon S3 bucket. If it is
            // different from the Region defined for the default user,
            // pass the Region to the constructor for the client. For
            // example: new AmazonS3Client(RegionEndpoint.USEast1);

            // If using the Region us-east-1, and server-side encryption with AWS
 KMS, you must specify Signature Version 4.
            // Region us-east-1 defaults to Signature Version 2 unless explicitly
 set to Version 4 as shown below.
            // For more details, see https://docs.aws.amazon.com/AmazonS3/latest/
userguide/UsingAWSSDK.html#specify-signature-version
            // and https://docs.aws.amazon.com/sdkfornet/v3/apidocs/items/Amazon/
TAWSConfigsS3.html
            AWSConfigsS3.UseSignatureVersion4 = true;
            IAmazonS3 s3Client = new AmazonS3Client(RegionEndpoint.USEast1);

            string urlString = GeneratePresignedURL(s3Client, bucketName, objectKey,
 timeoutDuration);
```

```
            Console.WriteLine($"The generated URL is: {urlString}.");
        }

        /// <summary>
        /// Generate a presigned URL that can be used to access the file named
        /// in the objectKey parameter for the amount of time specified in the
        /// duration parameter.
        /// </summary>
        /// <param name="client">An initialized S3 client object used to call
        /// the GetPresignedUrl method.</param>
        /// <param name="bucketName">The name of the S3 bucket containing the
        /// object for which to create the presigned URL.</param>
        /// <param name="objectKey">The name of the object to access with the
        /// presigned URL.</param>
        /// <param name="duration">The length of time for which the presigned
        /// URL will be valid.</param>
        /// <returns>A string representing the generated presigned URL.</returns>
        public static string GeneratePresignedURL(IAmazonS3 client, string
bucketName, string objectKey, double duration)
        {
            string urlString = string.Empty;
            try
            {
                var request = new GetPreSignedUrlRequest()
                {
                    BucketName = bucketName,
                    Key = objectKey,
                    Expires = DateTime.UtcNow.AddHours(duration),
                };
                urlString = client.GetPreSignedURL(request);
            }
            catch (AmazonS3Exception ex)
            {
                Console.WriteLine($"Error:'{ex.Message}'");
            }

            return urlString;
        }
    }
```

署名済み URL を生成し、その URL を使用してアップロードを実行します。

```
    using System;
    using System.IO;
    using System.Net.Http;
    using System.Threading.Tasks;
    using Amazon;
    using Amazon.S3;
    using Amazon.S3.Model;

    /// <summary>
    /// This example shows how to upload an object to an Amazon Simple Storage
    /// Service (Amazon S3) bucket using a presigned URL. The code first
    /// creates a presigned URL and then uses it to upload an object to an
    /// Amazon S3 bucket using that URL.
    /// </summary>
    public class UploadUsingPresignedURL
    {
        private static HttpClient httpClient = new HttpClient();

        public static async Task Main()
        {
            string bucketName = "doc-example-bucket";
            string keyName = "samplefile.txt";
            string filePath = $"source\\{keyName}";

            // Specify how long the signed URL will be valid in hours.
            double timeoutDuration = 12;

            // Specify the AWS Region of your Amazon S3 bucket. If it is
            // different from the Region defined for the default user,
            // pass the Region to the constructor for the client. For
            // example: new AmazonS3Client(RegionEndpoint.USEast1);

            // If using the Region us-east-1, and server-side encryption with AWS
 KMS, you must specify Signature Version 4.
            // Region us-east-1 defaults to Signature Version 2 unless explicitly
 set to Version 4 as shown below.
            // For more details, see https://docs.aws.amazon.com/AmazonS3/latest/
userguide/UsingAWSSDK.html#specify-signature-version
            // and https://docs.aws.amazon.com/sdkfornet/v3/apidocs/items/Amazon/
TAWSConfigsS3.html
            AWSConfigsS3.UseSignatureVersion4 = true;
            IAmazonS3 client = new AmazonS3Client(RegionEndpoint.USEast1);
```

```
            var url = GeneratePreSignedURL(client, bucketName, keyName,
timeoutDuration);
            var success = await UploadObject(filePath, url);

            if (success)
            {
                Console.WriteLine("Upload succeeded.");
            }
            else
            {
                Console.WriteLine("Upload failed.");
            }
        }

        /// <summary>
        /// Uploads an object to an Amazon S3 bucket using the presigned URL passed
in
        /// the url parameter.
        /// </summary>
        /// <param name="filePath">The path (including file name) to the local
        /// file you want to upload.</param>
        /// <param name="url">The presigned URL that will be used to upload the
        /// file to the Amazon S3 bucket.</param>
        /// <returns>A Boolean value indicating the success or failure of the
        /// operation, based on the HttpWebResponse.</returns>
        public static async Task<bool> UploadObject(string filePath, string url)
        {
            using var streamContent = new StreamContent(
                new FileStream(filePath, FileMode.Open, FileAccess.Read));

            var response = await httpClient.PutAsync(url, streamContent);
            return response.IsSuccessStatusCode;
        }

        /// <summary>
        /// Generates a presigned URL which will be used to upload an object to
        /// an Amazon S3 bucket.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used to
call
        /// GetPreSignedURL.</param>
        /// <param name="bucketName">The name of the Amazon S3 bucket to which the
        /// presigned URL will point.</param>
```

```
        /// <param name="objectKey">The name of the file that will be uploaded.</
param>
        /// <param name="duration">How long (in hours) the presigned URL will
        /// be valid.</param>
        /// <returns>The generated URL.</returns>
        public static string GeneratePreSignedURL(
            IAmazonS3 client,
            string bucketName,
            string objectKey,
            double duration)
        {
            var request = new GetPreSignedUrlRequest
            {
                BucketName = bucketName,
                Key = objectKey,
                Verb = HttpVerb.PUT,
                Expires = DateTime.UtcNow.AddHours(duration),
            };

            string url = client.GetPreSignedURL(request);
            return url;
        }
    }
```

バケットとオブジェクトの使用を開始する

次のコードサンプルは、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。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    public class S3_Basics
    {
        public static async Task Main()
        {
            // Create an Amazon S3 client object. The constructor uses the
            // default user installed on the system. To work with Amazon S3
            // features in a different AWS Region, pass the AWS Region as a
            // parameter to the client constructor.
            IAmazonS3 client = new AmazonS3Client();
            string bucketName = string.Empty;
            string filePath = string.Empty;
            string keyName = string.Empty;

            var sepBar = new string('-', Console.WindowWidth);

            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);

            // Create a bucket.
            Console.WriteLine($"\n{sepBar}");
            Console.WriteLine("\nCreate a new Amazon S3 bucket.\n");
            Console.WriteLine(sepBar);

            Console.Write("Please enter a name for the new bucket: ");
            bucketName = Console.ReadLine();
```

```
            var success = await S3Bucket.CreateBucketAsync(client, 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);

            // 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;
                }
            }

            // Get the file name from the full path.
            keyName = Path.GetFileName(filePath);

            success = await S3Bucket.UploadFileAsync(client, bucketName, keyName,
filePath);

            if (success)
            {
                Console.WriteLine($"Successfully uploaded {keyName} from {filePath}
to {bucketName}.\n");
            }
            else
            {
                Console.WriteLine($"Could not upload {keyName}.\n");
```

```
            }

            // Set the file path to an empty string to avoid overwriting the
            // file we just uploaded to the bucket.
            filePath = string.Empty;

            // Now get a new location where we can save the file.
            while (string.IsNullOrEmpty(filePath))
            {
                // First get the path to which the file will be downloaded.
                Console.Write("Please enter the path where the file will be
downloaded: ");
                filePath = Console.ReadLine();

                // Confirm that the file exists on the local computer.
                if (File.Exists($"{filePath}\\{keyName}"))
                {
                    Console.WriteLine($"Sorry, the file already exists in that
location.\n");
                    filePath = string.Empty;
                }
            }

            // Download an object from a bucket.
            success = await S3Bucket.DownloadObjectFromBucketAsync(client,
bucketName, keyName, filePath);

            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;

            while (string.IsNullOrEmpty(folderName))
            {
                Console.Write("Please enter the name of the folder to copy your
object to: ");
                folderName = Console.ReadLine();
```

```
            }

            while (string.IsNullOrEmpty(keyName))
            {
                // Get the name to give to the object once uploaded.
                Console.Write("Enter the name of the object to copy: ");
                keyName = Console.ReadLine();
            }

            await S3Bucket.CopyObjectInBucketAsync(client, bucketName, keyName,
folderName);

            // List the objects in the bucket.
            await S3Bucket.ListBucketContentsAsync(client, bucketName);

            // Delete the contents of the bucket.
            await S3Bucket.DeleteBucketContentsAsync(client, bucketName);

            // Deleting the bucket too quickly after deleting its contents will
            // cause an error that the bucket isn't empty. So...
            Console.WriteLine("Press <Enter> when you are ready to delete the
bucket.");
            _ = Console.ReadLine();

            // Delete the bucket.
            await S3Bucket.DeleteBucketAsync(client, bucketName);
        }
    }
```

- 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

暗号化の開始方法

次のコード例は、Amazon S3 オブジェクトの暗号化を開始する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
using System;
using System.IO;
using System.Security.Cryptography;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example shows how to apply client encryption to an object in an
/// Amazon Simple Storage Service (Amazon S3) bucket.
/// </summary>
public class SSEClientEncryption
{
    public static async Task Main()
    {
        string bucketName = "doc-example-bucket";
        string keyName = "exampleobject.txt";
        string copyTargetKeyName = "examplecopy.txt";

        // If the AWS Region defined for your default user is different
        // from the Region where your Amazon S3 bucket is located,
        // pass the Region name to the Amazon S3 client object's constructor.
        // For example: RegionEndpoint.USWest2.
        IAmazonS3 client = new AmazonS3Client();

        try
        {
            // Create an encryption key.
            Aes aesEncryption = Aes.Create();
            aesEncryption.KeySize = 256;
            aesEncryption.GenerateKey();
```

```
                string base64Key = Convert.ToBase64String(aesEncryption.Key);

                // Upload the object.
                PutObjectRequest putObjectRequest = await UploadObjectAsync(client,
 bucketName, keyName, base64Key);

                // Download the object and verify that its contents match what you
 uploaded.
                await DownloadObjectAsync(client, bucketName, keyName, base64Key,
 putObjectRequest);

                // Get object metadata and verify that the object uses AES-256
 encryption.
                await GetObjectMetadataAsync(client, bucketName, keyName,
 base64Key);

                // Copy both the source and target objects using server-side
 encryption with
                // an encryption key.
                await CopyObjectAsync(client, bucketName, keyName,
 copyTargetKeyName, aesEncryption, base64Key);
            }
            catch (AmazonS3Exception ex)
            {
                Console.WriteLine($"Error: {ex.Message}");
            }
        }

        /// <summary>
        /// Uploads an object to an Amazon S3 bucket.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used to
 call
        /// PutObjectAsync.</param>
        /// <param name="bucketName">The name of the Amazon S3 bucket to which the
        /// object will be uploaded.</param>
        /// <param name="keyName">The name of the object to upload to the Amazon S3
        /// bucket.</param>
        /// <param name="base64Key">The encryption key.</param>
        /// <returns>The PutObjectRequest object for use by DownloadObjectAsync.</
returns>
        public static async Task<PutObjectRequest> UploadObjectAsync(
            IAmazonS3 client,
            string bucketName,
```

```
            string keyName,
            string base64Key)
        {
            PutObjectRequest putObjectRequest = new PutObjectRequest
            {
                BucketName = bucketName,
                Key = keyName,
                ContentBody = "sample text",
                ServerSideEncryptionCustomerMethod =
 ServerSideEncryptionCustomerMethod.AES256,
                ServerSideEncryptionCustomerProvidedKey = base64Key,
            };
            PutObjectResponse putObjectResponse = await
 client.PutObjectAsync(putObjectRequest);
            return putObjectRequest;
        }

        /// <summary>
        /// Downloads an encrypted object from an Amazon S3 bucket.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used to
 call
        /// GetObjectAsync.</param>
        /// <param name="bucketName">The name of the Amazon S3 bucket where the
 object
        /// is located.</param>
        /// <param name="keyName">The name of the Amazon S3 object to download.</
param>
        /// <param name="base64Key">The encryption key used to encrypt the
        /// object.</param>
        /// <param name="putObjectRequest">The PutObjectRequest used to upload
        /// the object.</param>
        public static async Task DownloadObjectAsync(
            IAmazonS3 client,
            string bucketName,
            string keyName,
            string base64Key,
            PutObjectRequest putObjectRequest)
        {
            GetObjectRequest getObjectRequest = new GetObjectRequest
            {
                BucketName = bucketName,
                Key = keyName,
```

```
                // Provide encryption information for the object stored in Amazon
S3.
                ServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
                ServerSideEncryptionCustomerProvidedKey = base64Key,
            };

            using (GetObjectResponse getResponse = await
client.GetObjectAsync(getObjectRequest))
            using (StreamReader reader = new
StreamReader(getResponse.ResponseStream))
            {
                string content = reader.ReadToEnd();
                if (string.Compare(putObjectRequest.ContentBody, content) == 0)
                {
                    Console.WriteLine("Object content is same as we uploaded");
                }
                else
                {
                    Console.WriteLine("Error...Object content is not same.");
                }

                if (getResponse.ServerSideEncryptionCustomerMethod ==
ServerSideEncryptionCustomerMethod.AES256)
                {
                    Console.WriteLine("Object encryption method is AES256, same as
we set");
                }
                else
                {
                    Console.WriteLine("Error...Object encryption method is not the
same as AES256 we set");
                }
            }
        }

        /// <summary>
        /// Retrieves the metadata associated with an Amazon S3 object.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used
        /// to call GetObjectMetadataAsync.</param>
        /// <param name="bucketName">The name of the Amazon S3 bucket containing the
        /// object for which we want to retrieve metadata.</param>
        /// <param name="keyName">The name of the object for which we wish to
```

```
        /// retrieve the metadata.</param>
        /// <param name="base64Key">The encryption key associated with the
        /// object.</param>
        public static async Task GetObjectMetadataAsync(
            IAmazonS3 client,
            string bucketName,
            string keyName,
            string base64Key)
        {
            GetObjectMetadataRequest getObjectMetadataRequest = new
GetObjectMetadataRequest
            {
                BucketName = bucketName,
                Key = keyName,

                // The object stored in Amazon S3 is encrypted, so provide the
necessary encryption information.
                ServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
                ServerSideEncryptionCustomerProvidedKey = base64Key,
            };

            GetObjectMetadataResponse getObjectMetadataResponse = await
client.GetObjectMetadataAsync(getObjectMetadataRequest);
            Console.WriteLine("The object metadata show encryption method used is:
{0}", getObjectMetadataResponse.ServerSideEncryptionCustomerMethod);
        }

        /// <summary>
        /// Copies an encrypted object from one Amazon S3 bucket to another.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used to
call
        /// CopyObjectAsync.</param>
        /// <param name="bucketName">The Amazon S3 bucket containing the object
        /// to copy.</param>
        /// <param name="keyName">The name of the object to copy.</param>
        /// <param name="copyTargetKeyName">The Amazon S3 bucket to which the object
        /// will be copied.</param>
        /// <param name="aesEncryption">The encryption type to use.</param>
        /// <param name="base64Key">The encryption key to use.</param>
        public static async Task CopyObjectAsync(
            IAmazonS3 client,
            string bucketName,
```

```
            string keyName,
            string copyTargetKeyName,
            Aes aesEncryption,
            string base64Key)
        {
            aesEncryption.GenerateKey();
            string copyBase64Key = Convert.ToBase64String(aesEncryption.Key);

            CopyObjectRequest copyRequest = new CopyObjectRequest
            {
                SourceBucket = bucketName,
                SourceKey = keyName,
                DestinationBucket = bucketName,
                DestinationKey = copyTargetKeyName,

                // Information about the source object's encryption.
                CopySourceServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
                CopySourceServerSideEncryptionCustomerProvidedKey = base64Key,

                // Information about the target object's encryption.
                ServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
                ServerSideEncryptionCustomerProvidedKey = copyBase64Key,
            };
            await client.CopyObjectAsync(copyRequest);
        }
    }
```

- 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
  - GetObject
  - GetObjectMetadata

タグの使用開始

次のコード例は、Amazon S3 オブジェクトのタグの使用を開始する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example shows how to work with tags in Amazon Simple Storage
/// Service (Amazon S3) objects.
/// </summary>
public class ObjectTag
{
    public static async Task Main()
    {
        string bucketName = "doc-example-bucket";
        string keyName = "newobject.txt";
        string filePath = @"*** file path ***";

        // Specify your bucket region (an example region is shown).
        RegionEndpoint bucketRegion = RegionEndpoint.USWest2;

        var client = new AmazonS3Client(bucketRegion);
        await PutObjectsWithTagsAsync(client, bucketName, keyName, filePath);
    }

    /// <summary>
    /// This method uploads an object with tags. It then shows the tag
    /// values, changes the tags, and shows the new tags.
    /// </summary>
    /// <param name="client">The Initialized Amazon S3 client object used
    /// to call the methods to create and change an objects tags.</param>
    /// <param name="bucketName">A string representing the name of the
    /// bucket where the object will be stored.</param>
```

```
        /// <param name="keyName">A string representing the key name of the
        /// object to be tagged.</param>
        /// <param name="filePath">The directory location and file name of the
        /// object to be uploaded to the Amazon S3 bucket.</param>
        public static async Task PutObjectsWithTagsAsync(IAmazonS3 client, string
bucketName, string keyName, string filePath)
        {
            try
            {
                // Create an object with tags.
                var putRequest = new PutObjectRequest
                {
                    BucketName = bucketName,
                    Key = keyName,
                    FilePath = filePath,
                    TagSet = new List<Tag>
                    {
                        new Tag { Key = "Keyx1", Value = "Value1" },
                        new Tag { Key = "Keyx2", Value = "Value2" },
                    },
                };

                PutObjectResponse response = await
client.PutObjectAsync(putRequest);

                // Now retrieve the new object's tags.
                GetObjectTaggingRequest getTagsRequest = new
GetObjectTaggingRequest()
                {
                    BucketName = bucketName,
                    Key = keyName,
                };

                GetObjectTaggingResponse objectTags = await
client.GetObjectTaggingAsync(getTagsRequest);

                // Display the tag values.
                objectTags.Tagging
                    .ForEach(t => Console.WriteLine($"Key: {t.Key}, Value:
{t.Value}"));

                Tagging newTagSet = new Tagging()
                {
                    TagSet = new List<Tag>
```

```
                    {
                        new Tag { Key = "Key3", Value = "Value3" },
                        new Tag { Key = "Key4", Value = "Value4" },
                    },
                };

                PutObjectTaggingRequest putObjTagsRequest = new
PutObjectTaggingRequest()
                {
                    BucketName = bucketName,
                    Key = keyName,
                    Tagging = newTagSet,
                };

                PutObjectTaggingResponse response2 = await
client.PutObjectTaggingAsync(putObjTagsRequest);

                // Retrieve the tags again and show the values.
                GetObjectTaggingRequest getTagsRequest2 = new
GetObjectTaggingRequest()
                {
                    BucketName = bucketName,
                    Key = keyName,
                };
                GetObjectTaggingResponse objectTags2 = await
client.GetObjectTaggingAsync(getTagsRequest2);

                objectTags2.Tagging
                    .ForEach(t => Console.WriteLine($"Key: {t.Key}, Value:
{t.Value}"));
            }
            catch (AmazonS3Exception ex)
            {
                Console.WriteLine(
                        $"Error: '{ex.Message}'");
            }
        }
    }
```

- API の詳細については、「 API リファレンスGetObjectTagging」の「」を参照してください。 AWS SDK for .NET

オブジェクトのリーガルホールド設定を取得する

次のコード例は、S3 バケットのリーガルホールド設定を取得する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get the legal hold details for an S3 object.
    /// </summary>
    /// <param name="bucketName">The bucket of the object.</param>
    /// <param name="objectKey">The object key.</param>
    /// <returns>The object legal hold details.</returns>
    public async Task<ObjectLockLegalHold> GetObjectLegalHold(string bucketName,
        string objectKey)
    {
        try
        {
            var request = new GetObjectLegalHoldRequest()
            {
                BucketName = bucketName,
                Key = objectKey
            };

            var response = await _amazonS3.GetObjectLegalHoldAsync(request);
            Console.WriteLine($"\tObject legal hold for {objectKey} in {bucketName}:
" +
                              $"\n\tStatus: {response.LegalHold.Status}");
            return response.LegalHold;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"\tUnable to fetch legal hold: '{ex.Message}'");
            return new ObjectLockLegalHold();
        }
    }
```

- API の詳細については、「 API リファレンスGetObjectLegalHold」の「」を参照してください。 AWS SDK for .NET

Amazon S3 オブジェクトをロックする

次のコード例は、S3 オブジェクトロック機能を操作する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon S3 オブジェクトロック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.S3;
using Amazon.S3.Model;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;

namespace S3ObjectLockScenario;

public static class S3ObjectLockWorkflow
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.

    This .NET example performs the following tasks:
        1. Create test Amazon Simple Storage Service (S3) buckets with different
 lock policies.
        2. Upload sample objects to each bucket.
        3. Set some Legal Hold and Retention Periods on objects and buckets.
        4. Investigate lock policies by viewing settings or attempting to delete or
 overwrite objects.
```

```
        5. Clean up objects and buckets.
  */

   public static S3ActionsWrapper _s3ActionsWrapper = null!;
   public static IConfiguration _configuration = null!;
   private static string _resourcePrefix = null!;
   private static string noLockBucketName = null!;
   private static string lockEnabledBucketName = null!;
   private static string retentionAfterCreationBucketName = null!;
   private static List<string> bucketNames = new List<string>();
   private static List<string> fileNames = new List<string>();

   public 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<IAmazonS3>()
                   .AddTransient<S3ActionsWrapper>()
           )
           .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();

       ConfigurationSetup();

       ServicesSetup(host);

       try
       {
           Console.WriteLine(new string('-', 80));
           Console.WriteLine("Welcome to the Amazon Simple Storage Service (S3)
Object Locking Workflow Scenario.");
           Console.WriteLine(new string('-', 80));
```

```
            await Setup(true);

            await DemoActionChoices();

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Cleaning up resources.");
            Console.WriteLine(new string('-', 80));
            await Cleanup(true);

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Amazon S3 Object Locking Workflow is complete.");
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine($"There was a problem: {ex.Message}");
            await Cleanup(true);
            Console.WriteLine(new string('-', 80));
        }
    }

    /// <summary>
    /// Populate the services for use within the console application.
    /// </summary>
    /// <param name="host">The services host.</param>
    private static void ServicesSetup(IHost host)
    {
        _s3ActionsWrapper = host.Services.GetRequiredService<S3ActionsWrapper>();
    }

    /// <summary>
    /// Any setup operations needed.
    /// </summary>
    public static void ConfigurationSetup()
    {
        _resourcePrefix = _configuration["resourcePrefix"] ?? "dotnet-example";

        noLockBucketName = _resourcePrefix + "-no-lock";
        lockEnabledBucketName = _resourcePrefix + "-lock-enabled";
        retentionAfterCreationBucketName = _resourcePrefix + "-retention-after-
creation";

        bucketNames.Add(noLockBucketName);
```

```
        bucketNames.Add(lockEnabledBucketName);
        bucketNames.Add(retentionAfterCreationBucketName);
    }

    // <summary>
    /// Deploy necessary resources for the scenario.
    /// </summary>
    /// <param name="interactive">True to run as interactive.</param>
    /// <returns>True if successful.</returns>
    public static async Task<bool> Setup(bool interactive)
    {
        Console.WriteLine(
            "\nFor this workflow, we will use the AWS SDK for .NET to create several
S3\n" +
            "buckets and files to demonstrate working with S3 locking features.\n");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Press Enter when you are ready to start.");
        if (interactive)
            Console.ReadLine();

        Console.WriteLine("\nS3 buckets can be created either with or without object
lock enabled.");
        await _s3ActionsWrapper.CreateBucketWithObjectLock(noLockBucketName, false);
        await _s3ActionsWrapper.CreateBucketWithObjectLock(lockEnabledBucketName,
true);
        await
_s3ActionsWrapper.CreateBucketWithObjectLock(retentionAfterCreationBucketName,
false);

        Console.WriteLine("Press Enter to continue.");
        if (interactive)
            Console.ReadLine();

        Console.WriteLine("\nA bucket can be configured to use object locking with a
default retention period.");
        await
_s3ActionsWrapper.ModifyBucketDefaultRetention(retentionAfterCreationBucketName,
true,
            ObjectLockRetentionMode.Governance, DateTime.UtcNow.AddDays(1));

        Console.WriteLine("Press Enter to continue.");
        if (interactive)
            Console.ReadLine();
```

```
        Console.WriteLine("\nObject lock policies can also be added to existing
buckets.");
        await _s3ActionsWrapper.EnableObjectLockOnBucket(lockEnabledBucketName);

        Console.WriteLine("Press Enter to continue.");
        if (interactive)
            Console.ReadLine();

        // Upload some files to the buckets.
        Console.WriteLine("\nNow let's add some test files:");
        var fileName = _configuration["exampleFileName"] ?? "exampleFile.txt";
        int fileCount = 2;
        // Create the file if it does not already exist.
        if (!File.Exists(fileName))
        {
            await using StreamWriter sw = File.CreateText(fileName);
            await sw.WriteLineAsync(
                "This is a sample file for uploading to a bucket.");
        }

        foreach (var bucketName in bucketNames)
        {
            for (int i = 0; i < fileCount; i++)
            {
                var numberedFileName = Path.GetFileNameWithoutExtension(fileName) +
i + Path.GetExtension(fileName);
                fileNames.Add(numberedFileName);
                await _s3ActionsWrapper.UploadFileAsync(bucketName,
numberedFileName, fileName);
            }
        }
        Console.WriteLine("Press Enter to continue.");
        if (interactive)
            Console.ReadLine();

        if (!interactive)
            return true;
        Console.WriteLine("\nNow we can set some object lock policies on individual
files:");
        foreach (var bucketName in bucketNames)
        {
            for (int i = 0; i < fileNames.Count; i++)
            {
```

```
                // No modifications to the objects in the first bucket.
                if (bucketName != bucketNames[0])
                {
                    var exampleFileName = fileNames[i];
                    switch (i)
                    {
                        case 0:
                            {
                                var question =
                                    $"\nWould you like to add a legal hold to
{exampleFileName} in {bucketName}? (y/n)";
                                if (GetYesNoResponse(question))
                                {
                                    // Set a legal hold.
                                    await
_s3ActionsWrapper.ModifyObjectLegalHold(bucketName, exampleFileName,
ObjectLockLegalHoldStatus.On);

                                }
                                break;
                            }
                        case 1:
                            {
                                var question =
                                    $"\nWould you like to add a 1 day Governance
retention period to {exampleFileName} in {bucketName}? (y/n)" +
                                    "\nReminder: Only a user with the
s3:BypassGovernanceRetention permission will be able to delete this file or its
bucket until the retention period has expired.";
                                if (GetYesNoResponse(question))
                                {
                                    // Set a Governance mode retention period for 1
day.
                                    await
_s3ActionsWrapper.ModifyObjectRetentionPeriod(
                                        bucketName, exampleFileName,
                                        ObjectLockRetentionMode.Governance,
                                        DateTime.UtcNow.AddDays(1));
                                }
                                break;
                            }
                    }
                }
            }
```

```
        }
        Console.WriteLine(new string('-', 80));
        return true;
    }

    // <summary>
    /// List all of the current buckets and objects.
    /// </summary>
    /// <param name="interactive">True to run as interactive.</param>
    /// <returns>The list of buckets and objects.</returns>
    public static async Task<List<S3ObjectVersion>> ListBucketsAndObjects(bool
interactive)
    {
        var allObjects = new List<S3ObjectVersion>();
        foreach (var bucketName in bucketNames)
        {
            var objectsInBucket = await
_s3ActionsWrapper.ListBucketObjectsAndVersions(bucketName);
            foreach (var objectKey in objectsInBucket.Versions)
            {
                allObjects.Add(objectKey);
            }
        }

        if (interactive)
        {
            Console.WriteLine("\nCurrent buckets and objects:\n");
            int i = 0;
            foreach (var bucketObject in allObjects)
            {
                i++;
                Console.WriteLine(
                    $"{i}: {bucketObject.Key} \n\tBucket:
{bucketObject.BucketName}\n\tVersion: {bucketObject.VersionId}");
            }
        }

        return allObjects;
    }

    /// <summary>
    /// Present the user with the demo action choices.
    /// </summary>
    /// <returns>Async task.</returns>
```

```
    public static async Task<bool> DemoActionChoices()
    {
        var choices = new string[]{
            "List all files in buckets.",
            "Attempt to delete a file.",
            "Attempt to delete a file with retention period bypass.",
            "Attempt to overwrite a file.",
            "View the object and bucket retention settings for a file.",
            "View the legal hold settings for a file.",
            "Finish the workflow."};

        var choice = 0;
        // Keep asking the user until they choose to move on.
        while (choice != 6)
        {
            Console.WriteLine(new string('-', 80));
            choice = GetChoiceResponse(
                "\nExplore the S3 locking features by selecting one of the following
choices:"
                , choices);
            Console.WriteLine(new string('-', 80));
            switch (choice)
            {
                case 0:
                    {
                        await ListBucketsAndObjects(true);
                        break;
                    }
                case 1:
                    {
                        Console.WriteLine("\nEnter the number of the object to
delete:");
                        var allFiles = await ListBucketsAndObjects(true);
                        var fileChoice = GetChoiceResponse(null, allFiles.Select(f
=> f.Key).ToArray());
                        await
_s3ActionsWrapper.DeleteObjectFromBucket(allFiles[fileChoice].BucketName,
allFiles[fileChoice].Key, false, allFiles[fileChoice].VersionId);
                        break;
                    }
                case 2:
                    {
                        Console.WriteLine("\nEnter the number of the object to
delete:");
```

```
                        var allFiles = await ListBucketsAndObjects(true);
                        var fileChoice = GetChoiceResponse(null, allFiles.Select(f
=> f.Key).ToArray());
                        await
_s3ActionsWrapper.DeleteObjectFromBucket(allFiles[fileChoice].BucketName,
allFiles[fileChoice].Key, true, allFiles[fileChoice].VersionId);
                        break;
                    }
                case 3:
                    {
                        var allFiles = await ListBucketsAndObjects(true);
                        Console.WriteLine("\nEnter the number of the object to
overwrite:");
                        var fileChoice = GetChoiceResponse(null, allFiles.Select(f
=> f.Key).ToArray());
                        // Create the file if it does not already exist.
                        if (!File.Exists(allFiles[fileChoice].Key))
                        {
                            await using StreamWriter sw =
File.CreateText(allFiles[fileChoice].Key);
                            await sw.WriteLineAsync(
                                "This is a sample file for uploading to a bucket.");
                        }
                        await
_s3ActionsWrapper.UploadFileAsync(allFiles[fileChoice].BucketName,
allFiles[fileChoice].Key, allFiles[fileChoice].Key);
                        break;
                    }
                case 4:
                    {
                        var allFiles = await ListBucketsAndObjects(true);
                        Console.WriteLine("\nEnter the number of the object and
bucket to view:");
                        var fileChoice = GetChoiceResponse(null, allFiles.Select(f
=> f.Key).ToArray());
                        await
_s3ActionsWrapper.GetObjectRetention(allFiles[fileChoice].BucketName,
allFiles[fileChoice].Key);
                        await
_s3ActionsWrapper.GetBucketObjectLockConfiguration(allFiles[fileChoice].BucketName);
                        break;
                    }
                case 5:
                    {
```

```
                        var allFiles = await ListBucketsAndObjects(true);
                        Console.WriteLine("\nEnter the number of the object to
view:");
                        var fileChoice = GetChoiceResponse(null, allFiles.Select(f
=> f.Key).ToArray());
                        await
_s3ActionsWrapper.GetObjectLegalHold(allFiles[fileChoice].BucketName,
allFiles[fileChoice].Key);
                        break;
                    }
            }
        }
        return true;
    }

    // <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <param name="interactive">True to run as interactive.</param>
    /// <returns>True if successful.</returns>
    public static async Task<bool> Cleanup(bool interactive)
    {
        Console.WriteLine(new string('-', 80));

        if (!interactive || GetYesNoResponse("Do you want to clean up all files and
buckets? (y/n) "))
        {
            // Remove all locks and delete all buckets and objects.
            var allFiles = await ListBucketsAndObjects(false);
            foreach (var fileInfo in allFiles)
            {
                // Check for a legal hold.
                var legalHold = await
_s3ActionsWrapper.GetObjectLegalHold(fileInfo.BucketName, fileInfo.Key);
                if (legalHold?.Status?.Value == ObjectLockLegalHoldStatus.On)
                {
                    await
_s3ActionsWrapper.ModifyObjectLegalHold(fileInfo.BucketName, fileInfo.Key,
ObjectLockLegalHoldStatus.Off);
                }

                // Check for a retention period.
                var retention = await
_s3ActionsWrapper.GetObjectRetention(fileInfo.BucketName, fileInfo.Key);
```

```
                var hasRetentionPeriod = retention?.Mode ==
ObjectLockRetentionMode.Governance && retention.RetainUntilDate >
DateTime.UtcNow.Date;
                await _s3ActionsWrapper.DeleteObjectFromBucket(fileInfo.BucketName,
fileInfo.Key, hasRetentionPeriod, fileInfo.VersionId);
            }

            foreach (var bucketName in bucketNames)
            {
                await _s3ActionsWrapper.DeleteBucketByName(bucketName);
            }

        }
        else
        {
            Console.WriteLine(
                "Ok, we'll leave the resources intact.\n" +
                "Don't forget to delete them when you're done with them or you might
incur unexpected charges."
            );
        }

        Console.WriteLine(new string('-', 80));
        return true;
    }

    /// <summary>
    /// Helper method to 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;
    }

    /// <summary>
    /// Helper method to get a choice response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</param>
```

```
    /// <param name="choices">The choices to print on the console.</param>
    /// <returns>The index of the selected choice</returns>
    private static int GetChoiceResponse(string? question, string[] choices)
    {
        if (question != null)
        {
            Console.WriteLine(question);

            for (int i = 0; i < choices.Length; i++)
            {
                Console.WriteLine($"\t{i + 1}. {choices[i]}");
            }
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > choices.Length)
        {
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        return choiceNumber - 1;
    }
}
```

S3 関数のラッパークラス。

```
using System.Net;
using Amazon.S3;
using Amazon.S3.Model;
using Microsoft.Extensions.Configuration;

namespace S3ObjectLockScenario;

/// <summary>
/// Encapsulate the Amazon S3 operations.
/// </summary>
public class S3ActionsWrapper
{
    private readonly IAmazonS3 _amazonS3;
```

```
    /// <summary>
    /// Constructor for the S3ActionsWrapper.
    /// </summary>
    /// <param name="amazonS3">The injected S3 client.</param>
    public S3ActionsWrapper(IAmazonS3 amazonS3, IConfiguration configuration)
    {
        _amazonS3 = amazonS3;
    }

    /// <summary>
    /// Create a new Amazon S3 bucket with object lock actions.
    /// </summary>
    /// <param name="bucketName">The name of the bucket to create.</param>
    /// <param name="enableObjectLock">True to enable object lock on the bucket.</
param>
    /// <returns>True if successful.</returns>
    public async Task<bool> CreateBucketWithObjectLock(string bucketName, bool
 enableObjectLock)
    {
        Console.WriteLine($"\tCreating bucket {bucketName} with object lock
 {enableObjectLock}.");
        try
        {
            var request = new PutBucketRequest
            {
                BucketName = bucketName,
                UseClientRegion = true,
                ObjectLockEnabledForBucket = enableObjectLock,
            };

            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>
    /// Enable object lock on an existing bucket.
    /// </summary>
```

```
    /// <param name="bucketName">The name of the bucket to modify.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> EnableObjectLockOnBucket(string bucketName)
    {
        try
        {
            // First, enable Versioning on the bucket.
            await _amazonS3.PutBucketVersioningAsync(new
PutBucketVersioningRequest()
            {
                BucketName = bucketName,
                VersioningConfig = new S3BucketVersioningConfig()
                {
                    EnableMfaDelete = false,
                    Status = VersionStatus.Enabled
                }
            });

            var request = new PutObjectLockConfigurationRequest()
            {
                BucketName = bucketName,
                ObjectLockConfiguration = new ObjectLockConfiguration()
                {
                    ObjectLockEnabled = new ObjectLockEnabled("Enabled"),
                },
            };

            var response = await _amazonS3.PutObjectLockConfigurationAsync(request);
            Console.WriteLine($"\tAdded an object lock policy to bucket
{bucketName}.");
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error modifying object lock: '{ex.Message}'");
            return false;
        }
    }

    /// <summary>
    /// Set or modify a retention period on an object in an S3 bucket.
    /// </summary>
    /// <param name="bucketName">The bucket of the object.</param>
    /// <param name="objectKey">The key of the object.</param>
```

```
    /// <param name="retention">The retention mode.</param>
    /// <param name="retainUntilDate">The date retention expires.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> ModifyObjectRetentionPeriod(string bucketName,
        string objectKey, ObjectLockRetentionMode retention, DateTime
retainUntilDate)
    {
        try
        {
            var request = new PutObjectRetentionRequest()
            {
                BucketName = bucketName,
                Key = objectKey,
                Retention = new ObjectLockRetention()
                {
                    Mode = retention,
                    RetainUntilDate = retainUntilDate
                }
            };

            var response = await _amazonS3.PutObjectRetentionAsync(request);
            Console.WriteLine($"\tSet retention for {objectKey} in {bucketName}
until {retainUntilDate:d}.");
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"\tError modifying retention period:
'{ex.Message}'");
            return false;
        }
    }

    /// <summary>
    /// Set or modify a retention period on an S3 bucket.
    /// </summary>
    /// <param name="bucketName">The bucket to modify.</param>
    /// <param name="retention">The retention mode.</param>
    /// <param name="retainUntilDate">The date for retention until.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> ModifyBucketDefaultRetention(string bucketName, bool
enableObjectLock, ObjectLockRetentionMode retention, DateTime retainUntilDate)
    {
        var enabledString = enableObjectLock ? "Enabled" : "Disabled";
```

```
        var timeDifference = retainUntilDate.Subtract(DateTime.Now);
        try
        {
            // First, enable Versioning on the bucket.
            await _amazonS3.PutBucketVersioningAsync(new
PutBucketVersioningRequest()
            {
                BucketName = bucketName,
                VersioningConfig = new S3BucketVersioningConfig()
                {
                    EnableMfaDelete = false,
                    Status = VersionStatus.Enabled
                }
            });

            var request = new PutObjectLockConfigurationRequest()
            {
                BucketName = bucketName,
                ObjectLockConfiguration = new ObjectLockConfiguration()
                {
                    ObjectLockEnabled = new ObjectLockEnabled(enabledString),
                    Rule = new ObjectLockRule()
                    {
                        DefaultRetention = new DefaultRetention()
                        {
                            Mode = retention,
                            Days = timeDifference.Days // Can be specified in days
or years but not both.
                        }
                    }
                }
            };

            var response = await _amazonS3.PutObjectLockConfigurationAsync(request);
            Console.WriteLine($"\tAdded a default retention to bucket
{bucketName}.");
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"\tError modifying object lock: '{ex.Message}'");
            return false;
        }
    }
```

```
    /// <summary>
    /// Get the retention period for an S3 object.
    /// </summary>
    /// <param name="bucketName">The bucket of the object.</param>
    /// <param name="objectKey">The object key.</param>
    /// <returns>The object retention details.</returns>
    public async Task<ObjectLockRetention> GetObjectRetention(string bucketName,
        string objectKey)
    {
        try
        {
            var request = new GetObjectRetentionRequest()
            {
                BucketName = bucketName,
                Key = objectKey
            };

            var response = await _amazonS3.GetObjectRetentionAsync(request);
            Console.WriteLine($"\tObject retention for {objectKey} in {bucketName}:
" +
                              $"\n\t{response.Retention.Mode} until
{response.Retention.RetainUntilDate:d}.");
            return response.Retention;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"\tUnable to fetch object lock retention:
'{ex.Message}'");
            return new ObjectLockRetention();
        }
    }

    /// <summary>
    /// Set or modify a legal hold on an object in an S3 bucket.
    /// </summary>
    /// <param name="bucketName">The bucket of the object.</param>
    /// <param name="objectKey">The key of the object.</param>
    /// <param name="holdStatus">The On or Off status for the legal hold.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> ModifyObjectLegalHold(string bucketName,
        string objectKey, ObjectLockLegalHoldStatus holdStatus)
    {
        try
```

```
        {
            var request = new PutObjectLegalHoldRequest()
            {
                BucketName = bucketName,
                Key = objectKey,
                LegalHold = new ObjectLockLegalHold()
                {
                    Status = holdStatus
                }
            };

            var response = await _amazonS3.PutObjectLegalHoldAsync(request);
            Console.WriteLine($"\tModified legal hold for {objectKey} in
{bucketName}.");
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"\tError modifying legal hold: '{ex.Message}'");
            return false;
        }
    }

    /// <summary>
    /// Get the legal hold details for an S3 object.
    /// </summary>
    /// <param name="bucketName">The bucket of the object.</param>
    /// <param name="objectKey">The object key.</param>
    /// <returns>The object legal hold details.</returns>
    public async Task<ObjectLockLegalHold> GetObjectLegalHold(string bucketName,
        string objectKey)
    {
        try
        {
            var request = new GetObjectLegalHoldRequest()
            {
                BucketName = bucketName,
                Key = objectKey
            };

            var response = await _amazonS3.GetObjectLegalHoldAsync(request);
            Console.WriteLine($"\tObject legal hold for {objectKey} in {bucketName}:
" +
                              $"\n\tStatus: {response.LegalHold.Status}");
```

```
            return response.LegalHold;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"\tUnable to fetch legal hold: '{ex.Message}'");
            return new ObjectLockLegalHold();
        }
    }

    /// <summary>
    /// Get the object lock configuration details for an S3 bucket.
    /// </summary>
    /// <param name="bucketName">The bucket to get details.</param>
    /// <returns>The bucket's object lock configuration details.</returns>
    public async Task<ObjectLockConfiguration>
GetBucketObjectLockConfiguration(string bucketName)
    {
        try
        {
            var request = new GetObjectLockConfigurationRequest()
            {
                BucketName = bucketName
            };

            var response = await _amazonS3.GetObjectLockConfigurationAsync(request);
            Console.WriteLine($"\tBucket object lock config for {bucketName} in
{bucketName}: " +
                              $"\n\tEnabled:
{response.ObjectLockConfiguration.ObjectLockEnabled}" +
                              $"\n\tRule:
{response.ObjectLockConfiguration.Rule?.DefaultRetention}");

            return response.ObjectLockConfiguration;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"\tUnable to fetch object lock config:
'{ex.Message}'");
            return new ObjectLockConfiguration();
        }
    }

    /// <summary>
    /// Upload a file from the local computer to an Amazon S3 bucket.
```

```
    /// </summary>
    /// <param name="bucketName">The Amazon S3 bucket to use.</param>
    /// <param name="objectName">The object to upload.</param>
    /// <param name="filePath">The path, including file name, of the object to
upload.</param>
    /// <returns>True if success.<returns>
    public async Task<bool> UploadFileAsync(string bucketName, string objectName,
string filePath)
    {
        var request = new PutObjectRequest
        {
            BucketName = bucketName,
            Key = objectName,
            FilePath = filePath,
            ChecksumAlgorithm = ChecksumAlgorithm.SHA256
        };

        var response = await _amazonS3.PutObjectAsync(request);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"\tSuccessfully uploaded {objectName} to
{bucketName}.");
            return true;
        }
        else
        {
            Console.WriteLine($"\tCould not upload {objectName} to {bucketName}.");
            return false;
        }
    }

    /// <summary>
    /// List bucket objects and versions.
    /// </summary>
    /// <param name="bucketName">The Amazon S3 bucket to use.</param>
    /// <returns>The list of objects and versions.</returns>
    public async Task<ListVersionsResponse> ListBucketObjectsAndVersions(string
bucketName)
    {
        var request = new ListVersionsRequest()
        {
            BucketName = bucketName
        };
```

```
        var response = await _amazonS3.ListVersionsAsync(request);
        return response;
    }

    /// <summary>
    /// Delete an object from a specific bucket.
    /// </summary>
    /// <param name="bucketName">The Amazon S3 bucket to use.</param>
    /// <param name="objectKey">The key of the object to delete.</param>
    /// <param name="hasRetention">True if the object has retention settings.</
param>
    /// <param name="versionId">Optional versionId.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteObjectFromBucket(string bucketName, string
 objectKey, bool hasRetention, string? versionId = null)
    {
        try
        {
            var request = new DeleteObjectRequest()
            {
                BucketName = bucketName,
                Key = objectKey,
                VersionId = versionId,
            };
            if (hasRetention)
            {
                // Set the BypassGovernanceRetention header
                // if the file has retention settings.
                request.BypassGovernanceRetention = true;
            }
            await _amazonS3.DeleteObjectAsync(request);
            Console.WriteLine(
                $"Deleted {objectKey} in {bucketName}.");
            return true;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"\tUnable to delete object {objectKey} in bucket
 {bucketName}: " + ex.Message);
            return false;
        }
    }

    /// <summary>
```

```
    /// Delete a specific bucket.
    /// </summary>
    /// <param name="bucketName">The Amazon S3 bucket to use.</param>
    /// <param name="objectKey">The key of the object to delete.</param>
    /// <param name="versionId">Optional versionId.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteBucketByName(string bucketName)
    {
        try
        {
            var request = new DeleteBucketRequest() { BucketName = bucketName, };
            var response = await _amazonS3.DeleteBucketAsync(request);
            Console.WriteLine($"\tDelete for {bucketName} complete.");
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"\tUnable to delete bucket {bucketName}: " +
 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。
  - GetObjectLegalHold
  - GetObjectLockConfiguration
  - GetObjectRetention
  - PutObjectLegalHold
  - PutObjectLockConfiguration
  - PutObjectRetention

アクセスコントロールリスト (ACL) の管理

次のコード例は、Amazon S3 バケットのアクセスコントロールリスト (ACL) を管理する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example shows how to manage Amazon Simple Storage Service
/// (Amazon S3) access control lists (ACLs) to control Amazon S3 bucket
/// access.
/// </summary>
public class ManageACLs
{
    public static async Task Main()
    {
        string bucketName = "doc-example-bucket1";
        string newBucketName = "doc-example-bucket2";
        string keyName = "sample-object.txt";
        string emailAddress = "someone@example.com";

        // If the AWS Region where your bucket is located is different from
        // the Region defined for the default user, pass the Amazon S3 bucket's
        // name to the client constructor. It should look like this:
        // RegionEndpoint bucketRegion = RegionEndpoint.USEast1;
        IAmazonS3 client = new AmazonS3Client();

        await TestBucketObjectACLsAsync(client, bucketName, newBucketName,
keyName, emailAddress);
    }

    /// <summary>
    /// Creates a new Amazon S3 bucket with a canned ACL, then retrieves the ACL
    /// information and then adds a new ACL to one of the objects in the
    /// Amazon S3 bucket.
```

```
        /// </summary>
        /// <param name="client">The initialized Amazon S3 client object used to
call
        /// methods to create a bucket, get an ACL, and add a different ACL to
        /// one of the objects.</param>
        /// <param name="bucketName">A string representing the original Amazon S3
        /// bucket name.</param>
        /// <param name="newBucketName">A string representing the name of the
        /// new bucket that will be created.</param>
        /// <param name="keyName">A string representing the key name of an Amazon S3
        /// object for which we will change the ACL.</param>
        /// <param name="emailAddress">A string representing the email address
        /// belonging to the person to whom access to the Amazon S3 bucket will be
        /// granted.</param>
        public static async Task TestBucketObjectACLsAsync(
            IAmazonS3 client,
            string bucketName,
            string newBucketName,
            string keyName,
            string emailAddress)
        {
            try
            {
                // Create a new Amazon S3 bucket and specify canned ACL.
                var success = await CreateBucketWithCannedACLAsync(client,
newBucketName);

                // Get the ACL on a bucket.
                await GetBucketACLAsync(client, bucketName);

                // Add (replace) the ACL on an object in a bucket.
                await AddACLToExistingObjectAsync(client, bucketName, keyName,
emailAddress);
            }
            catch (AmazonS3Exception amazonS3Exception)
            {
                Console.WriteLine($"Exception: {amazonS3Exception.Message}");
            }
        }

        /// <summary>
        /// Creates a new Amazon S3 bucket with a canned ACL attached.
        /// </summary>
        /// <param name="client">The initialized client object used to call
```

```
        /// PutBucketAsync.</param>
        /// <param name="newBucketName">A string representing the name of the
        /// new Amazon S3 bucket.</param>
        /// <returns>Returns a boolean value indicating success or failure.</
returns>
        public static async Task<bool> CreateBucketWithCannedACLAsync(IAmazonS3
 client, string newBucketName)
        {
            var request = new PutBucketRequest()
            {
                BucketName = newBucketName,
                BucketRegion = S3Region.EUWest1,

                // Add a canned ACL.
                CannedACL = S3CannedACL.LogDeliveryWrite,
            };

            var response = await client.PutBucketAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }


        /// <summary>
        /// Retrieves the ACL associated with the Amazon S3 bucket name in the
        /// bucketName parameter.
        /// </summary>
        /// <param name="client">The initialized client object used to call
        /// PutBucketAsync.</param>
        /// <param name="bucketName">The Amazon S3 bucket for which we want to get
 the
        /// ACL list.</param>
        /// <returns>Returns an S3AccessControlList returned from the call to
        /// GetACLAsync.</returns>
        public static async Task<S3AccessControlList> GetBucketACLAsync(IAmazonS3
 client, string bucketName)
        {
            GetACLResponse response = await client.GetACLAsync(new GetACLRequest
            {
                BucketName = bucketName,
            });

            return response.AccessControlList;
        }
```

```
        /// <summary>
        /// Adds a new ACL to an existing object in the Amazon S3 bucket.
        /// </summary>
        /// <param name="client">The initialized client object used to call
        /// PutBucketAsync.</param>
        /// <param name="bucketName">A string representing the name of the Amazon S3
        /// bucket containing the object to which we want to apply a new ACL.</
param>
        /// <param name="keyName">A string representing the name of the object
        /// to which we want to apply the new ACL.</param>
        /// <param name="emailAddress">The email address of the person to whom
        /// we will be applying to whom access will be granted.</param>
        public static async Task AddACLToExistingObjectAsync(IAmazonS3 client,
 string bucketName, string keyName, string emailAddress)
        {
            // Retrieve the ACL for an object.
            GetACLResponse aclResponse = await client.GetACLAsync(new GetACLRequest
            {
                BucketName = bucketName,
                Key = keyName,
            });

            S3AccessControlList acl = aclResponse.AccessControlList;

            // Retrieve the owner.
            Owner owner = acl.Owner;

            // Clear existing grants.
            acl.Grants.Clear();

            // Add a grant to reset the owner's full permission
            // (the previous clear statement removed all permissions).
            var fullControlGrant = new S3Grant
            {
                Grantee = new S3Grantee { CanonicalUser = acl.Owner.Id },
            };
            acl.AddGrant(fullControlGrant.Grantee, S3Permission.FULL_CONTROL);

            // Specify email to identify grantee for granting permissions.
            var grantUsingEmail = new S3Grant
            {
                Grantee = new S3Grantee { EmailAddress = emailAddress },
```

```
                Permission = S3Permission.WRITE_ACP,
            };

            // Specify log delivery group as grantee.
            var grantLogDeliveryGroup = new S3Grant
            {
                Grantee = new S3Grantee { URI = "http://acs.amazonaws.com/groups/s3/
LogDelivery" },
                Permission = S3Permission.WRITE,
            };

            // Create a new ACL.
            var newAcl = new S3AccessControlList
            {
                Grants = new List<S3Grant> { grantUsingEmail,
 grantLogDeliveryGroup },
                Owner = owner,
            };

            // Set the new ACL. We're throwing away the response here.
            _ = await client.PutACLAsync(new PutACLRequest
            {
                BucketName = bucketName,
                Key = keyName,
                AccessControlList = newAcl,
            });
        }

    }
```

- 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。
  - GetBucketAcl
  - GetObjectAcl
  - PutBucketAcl
  - PutObjectAcl

マルチパートコピーを実行する

次のコード例は、Amazon S3 オブジェクトのマルチパートコピーを実行する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// This example shows how to perform a multi-part copy from one Amazon
/// Simple Storage Service (Amazon S3) bucket to another.
/// </summary>
public class MPUapiCopyObj
{
    private const string SourceBucket = "doc-example-bucket1";
    private const string TargetBucket = "doc-example-bucket2";
    private const string SourceObjectKey = "example.mov";
    private const string TargetObjectKey = "copied_video_file.mov";

    /// <summary>
    /// This method starts the multi-part upload.
    /// </summary>
    public static async Task Main()
    {
        var s3Client = new AmazonS3Client();
        Console.WriteLine("Copying object...");
        await MPUCopyObjectAsync(s3Client);
    }

    /// <summary>
    /// This method uses the passed client object to perform a multipart
    /// copy operation.
    /// </summary>
```

```
        /// <param name="client">An Amazon S3 client object that will be used
        /// to perform the copy.</param>
        public static async Task MPUCopyObjectAsync(AmazonS3Client client)
        {
            // Create a list to store the copy part responses.
            var copyResponses = new List<CopyPartResponse>();

            // Setup information required to initiate the multipart upload.
            var initiateRequest = new InitiateMultipartUploadRequest
            {
                BucketName = TargetBucket,
                Key = TargetObjectKey,
            };

            // Initiate the upload.
            InitiateMultipartUploadResponse initResponse =
                await client.InitiateMultipartUploadAsync(initiateRequest);

            // Save the upload ID.
            string uploadId = initResponse.UploadId;

            try
            {
                // Get the size of the object.
                var metadataRequest = new GetObjectMetadataRequest
                {
                    BucketName = SourceBucket,
                    Key = SourceObjectKey,
                };

                GetObjectMetadataResponse metadataResponse =
                    await client.GetObjectMetadataAsync(metadataRequest);
                var objectSize = metadataResponse.ContentLength; // Length in bytes.

                // Copy the parts.
                var partSize = 5 * (long)Math.Pow(2, 20); // Part size is 5 MB.

                long bytePosition = 0;
                for (int i = 1; bytePosition < objectSize; i++)
                {
                    var copyRequest = new CopyPartRequest
                    {
                        DestinationBucket = TargetBucket,
                        DestinationKey = TargetObjectKey,
```

```
                        SourceBucket = SourceBucket,
                        SourceKey = SourceObjectKey,
                        UploadId = uploadId,
                        FirstByte = bytePosition,
                        LastByte = bytePosition + partSize - 1 >= objectSize ?
objectSize - 1 : bytePosition + partSize - 1,
                        PartNumber = i,
                    };

                    copyResponses.Add(await client.CopyPartAsync(copyRequest));

                    bytePosition += partSize;
                }

                // Set up to complete the copy.
                var completeRequest = new CompleteMultipartUploadRequest
                {
                    BucketName = TargetBucket,
                    Key = TargetObjectKey,
                    UploadId = initResponse.UploadId,
                };
                completeRequest.AddPartETags(copyResponses);

                // Complete the copy.
                CompleteMultipartUploadResponse completeUploadResponse =
                    await client.CompleteMultipartUploadAsync(completeRequest);
            }
            catch (AmazonS3Exception e)
            {
                Console.WriteLine($"Error encountered on server.
Message:'{e.Message}' when writing an object");
            }
            catch (Exception e)
            {
                Console.WriteLine($"Unknown encountered on server.
Message:'{e.Message}' when writing an object");
            }
        }
    }
```

- 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。
  - CompleteMultipartUpload
  - CreateMultipartUpload
  - GetObjectMetadata
  - UploadPartCopy

大きなファイルをアップロードまたはダウンロードする

次のコード例は、Amazon S3 との間で大きなファイルをアップロードまたはダウンロードする方法を示しています。

詳細については、「マルチパートアップロードを使用したオブジェクトのアップロード」を参照してください。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon S3 TransferUtility S3 を使用して S3 バケットとの間でファイルを転送する関数を呼び出します。

```
global using System.Text;
global using Amazon.S3;
global using Amazon.S3.Model;
global using Amazon.S3.Transfer;
global using TransferUtilityBasics;



// This Amazon S3 client uses the default user credentials
// defined for this computer.
using Microsoft.Extensions.Configuration;

IAmazonS3 client = new AmazonS3Client();
var transferUtil = new TransferUtility(client);
```

```
IConfiguration _configuration;

_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();

// Edit the values in settings.json to use an S3 bucket and files that
// exist on your AWS account and on the local computer where you
// run this scenario.
var bucketName = _configuration["BucketName"];
var localPath =
 $"{Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData)}\
\TransferFolder";

DisplayInstructions();

PressEnter();

Console.WriteLine();

// Upload a single file to an S3 bucket.
DisplayTitle("Upload a single file");

var fileToUpload = _configuration["FileToUpload"];
Console.WriteLine($"Uploading {fileToUpload} to the S3 bucket, {bucketName}.");

var success = await TransferMethods.UploadSingleFileAsync(transferUtil, bucketName,
 fileToUpload, localPath);
if (success)
{
    Console.WriteLine($"Successfully uploaded the file, {fileToUpload} to
 {bucketName}.");
}

PressEnter();

// Upload a local directory to an S3 bucket.
DisplayTitle("Upload all files from a local directory");
Console.WriteLine("Upload all the files in a local folder to an S3 bucket.");
const string keyPrefix = "UploadFolder";
var uploadPath = $"{localPath}\\UploadFolder";
```

```
Console.WriteLine($"Uploading the files in {uploadPath} to {bucketName}");
DisplayTitle($"{uploadPath} files");
DisplayLocalFiles(uploadPath);
Console.WriteLine();

PressEnter();

success = await TransferMethods.UploadFullDirectoryAsync(transferUtil, bucketName,
 keyPrefix, uploadPath);
if (success)
{
    Console.WriteLine($"Successfully uploaded the files in {uploadPath} to
 {bucketName}.");
    Console.WriteLine($"{bucketName} currently contains the following files:");
    await DisplayBucketFiles(client, bucketName, keyPrefix);
    Console.WriteLine();
}

PressEnter();

// Download a single file from an S3 bucket.
DisplayTitle("Download a single file");
Console.WriteLine("Now we will download a single file from an S3 bucket.");

var keyName = _configuration["FileToDownload"];

Console.WriteLine($"Downloading {keyName} from {bucketName}.");

success = await TransferMethods.DownloadSingleFileAsync(transferUtil, bucketName,
 keyName, localPath);
if (success)
{
    Console.WriteLine("$Successfully downloaded the file, {keyName} from
 {bucketName}.");
}

PressEnter();

// Download the contents of a directory from an S3 bucket.
DisplayTitle("Download the contents of an S3 bucket");
var s3Path = _configuration["S3Path"];
var downloadPath = $"{localPath}\\{s3Path}";
```

```
Console.WriteLine($"Downloading the contents of {bucketName}\\{s3Path}");
Console.WriteLine($"{bucketName}\\{s3Path} contains the following files:");
await DisplayBucketFiles(client, bucketName, s3Path);
Console.WriteLine();

success = await TransferMethods.DownloadS3DirectoryAsync(transferUtil, bucketName,
 s3Path, downloadPath);
if (success)
{
    Console.WriteLine($"Downloaded the files in {bucketName} to {downloadPath}.");
    Console.WriteLine($"{downloadPath} now contains the following files:");
    DisplayLocalFiles(downloadPath);
}

Console.WriteLine("\nThe TransferUtility Basics application has completed.");
PressEnter();

// Displays the title for a section of the scenario.
static void DisplayTitle(string titleText)
{
    var sepBar = new string('-', Console.WindowWidth);

    Console.WriteLine(sepBar);
    Console.WriteLine(CenterText(titleText));
    Console.WriteLine(sepBar);
}

// Displays a description of the actions to be performed by the scenario.
static void DisplayInstructions()
{
    var sepBar = new string('-', Console.WindowWidth);

    DisplayTitle("Amazon S3 Transfer Utility Basics");
    Console.WriteLine("This program shows how to use the Amazon S3 Transfer
 Utility.");
    Console.WriteLine("It performs the following actions:");
    Console.WriteLine("\t1. Upload a single object to an S3 bucket.");
    Console.WriteLine("\t2. Upload an entire directory from the local computer to an
\n\t  S3 bucket.");
    Console.WriteLine("\t3. Download a single object from an S3 bucket.");
    Console.WriteLine("\t4. Download the objects in an S3 bucket to a local
 directory.");
    Console.WriteLine($"\n{sepBar}");
}
```

```
// Pauses the scenario.
static void PressEnter()
{
    Console.WriteLine("Press <Enter> to continue.");
    _ = Console.ReadLine();
    Console.WriteLine("\n");
}

// Returns the string textToCenter, padded on the left with spaces
// that center the text on the console display.
static string CenterText(string textToCenter)
{
    var centeredText = new StringBuilder();
    var screenWidth = Console.WindowWidth;
    centeredText.Append(new string(' ', (int)(screenWidth - textToCenter.Length) /
 2));
    centeredText.Append(textToCenter);
    return centeredText.ToString();
}

// Displays a list of file names included in the specified path.
static void DisplayLocalFiles(string localPath)
{
    var fileList = Directory.GetFiles(localPath);
    if (fileList.Length > 0)
    {
        foreach (var fileName in fileList)
        {
            Console.WriteLine(fileName);
        }
    }
}

// Displays a list of the files in the specified S3 bucket and prefix.
static async Task DisplayBucketFiles(IAmazonS3 client, string bucketName, string
 s3Path)
{
    ListObjectsV2Request request = new()
    {
        BucketName = bucketName,
        Prefix = s3Path,
        MaxKeys = 5,
    };
```

```
    var response = new ListObjectsV2Response();

    do
    {
        response = await client.ListObjectsV2Async(request);

        response.S3Objects
            .ForEach(obj => Console.WriteLine($"{obj.Key}"));

        // If the response is truncated, set the request ContinuationToken
        // from the NextContinuationToken property of the response.
        request.ContinuationToken = response.NextContinuationToken;
    } while (response.IsTruncated);
}
```

1 つのファイルをアップロードします。

```
        /// <summary>
        /// Uploads a single file from the local computer to an S3 bucket.
        /// </summary>
        /// <param name="transferUtil">The transfer initialized TransferUtility
        /// object.</param>
        /// <param name="bucketName">The name of the S3 bucket where the file
        /// will be stored.</param>
        /// <param name="fileName">The name of the file to upload.</param>
        /// <param name="localPath">The local path where the file is stored.</param>
        /// <returns>A boolean value indicating the success of the action.</returns>
        public static async Task<bool> UploadSingleFileAsync(
            TransferUtility transferUtil,
            string bucketName,
            string fileName,
            string localPath)
        {
            if (File.Exists($"{localPath}\\{fileName}"))
            {
                try
                {
                    await transferUtil.UploadAsync(new TransferUtilityUploadRequest
                    {
```

```
                        BucketName = bucketName,
                        Key = fileName,
                        FilePath = $"{localPath}\\{fileName}",
                    });

                    return true;
                }
                catch (AmazonS3Exception s3Ex)
                {
                    Console.WriteLine($"Could not upload {fileName} from {localPath}
 because:");
                    Console.WriteLine(s3Ex.Message);
                    return false;
                }
            }
            else
            {
                Console.WriteLine($"{fileName} does not exist in {localPath}");
                return false;
            }
        }
```

ローカルディレクトリ全体をアップロードします。

```
        /// <summary>
        /// Uploads all the files in a local directory to a directory in an S3
        /// bucket.
        /// </summary>
        /// <param name="transferUtil">The transfer initialized TransferUtility
        /// object.</param>
        /// <param name="bucketName">The name of the S3 bucket where the files
        /// will be stored.</param>
        /// <param name="keyPrefix">The key prefix is the S3 directory where
        /// the files will be stored.</param>
        /// <param name="localPath">The local directory that contains the files
        /// to be uploaded.</param>
        /// <returns>A Boolean value representing the success of the action.</
returns>
        public static async Task<bool> UploadFullDirectoryAsync(
            TransferUtility transferUtil,
            string bucketName,
```

```
            string keyPrefix,
            string localPath)
        {
            if (Directory.Exists(localPath))
            {
                try
                {
                    await transferUtil.UploadDirectoryAsync(new
TransferUtilityUploadDirectoryRequest
                    {
                        BucketName = bucketName,
                        KeyPrefix = keyPrefix,
                        Directory = localPath,
                    });

                    return true;
                }
                catch (AmazonS3Exception s3Ex)
                {
                    Console.WriteLine($"Can't upload the contents of {localPath}
because:");
                    Console.WriteLine(s3Ex?.Message);
                    return false;
                }
            }
            else
            {
                Console.WriteLine($"The directory {localPath} does not exist.");
                return false;
            }
        }
```

1 つのファイルをダウンロードします。

```
        /// <summary>
        /// Download a single file from an S3 bucket to the local computer.
        /// </summary>
        /// <param name="transferUtil">The transfer initialized TransferUtility
        /// object.</param>
        /// <param name="bucketName">The name of the S3 bucket containing the
```

```
        /// file to download.</param>
        /// <param name="keyName">The name of the file to download.</param>
        /// <param name="localPath">The path on the local computer where the
        /// downloaded file will be saved.</param>
        /// <returns>A Boolean value indicating the results of the action.</returns>
        public static async Task<bool> DownloadSingleFileAsync(
        TransferUtility transferUtil,
            string bucketName,
            string keyName,
            string localPath)
        {
            await transferUtil.DownloadAsync(new TransferUtilityDownloadRequest
            {
                BucketName = bucketName,
                Key = keyName,
                FilePath = $"{localPath}\\{keyName}",
            });

            return (File.Exists($"{localPath}\\{keyName}"));
        }
```

S3 バケットの内容をダウンロードします。

```
        /// <summary>
        /// Downloads the contents of a directory in an S3 bucket to a
        /// directory on the local computer.
        /// </summary>
        /// <param name="transferUtil">The transfer initialized TransferUtility
        /// object.</param>
        /// <param name="bucketName">The bucket containing the files to download.</
param>
        /// <param name="s3Path">The S3 directory where the files are located.</
param>
        /// <param name="localPath">The local path to which the files will be
        /// saved.</param>
        /// <returns>A Boolean value representing the success of the action.</
returns>
        public static async Task<bool> DownloadS3DirectoryAsync(
            TransferUtility transferUtil,
            string bucketName,
```

```
            string s3Path,
            string localPath)
        {
            int fileCount = 0;

            // If the directory doesn't exist, it will be created.
            if (Directory.Exists(s3Path))
            {
                var files = Directory.GetFiles(localPath);
                fileCount = files.Length;
            }

            await transferUtil.DownloadDirectoryAsync(new
TransferUtilityDownloadDirectoryRequest
            {
                BucketName = bucketName,
                LocalDirectory = localPath,
                S3Directory = s3Path,
            });

            if (Directory.Exists(localPath))
            {
                var files = Directory.GetFiles(localPath);
                if (files.Length > fileCount)
                {
                    return true;
                }

                // No change in the number of files. Assume
                // the download failed.
                return false;
            }

            // The local directory doesn't exist. No files
            // were downloaded.
            return false;
        }
```

を使用してアップロードの進行状況を追跡します TransferUtility。

```
    using System;
```

```
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Transfer;

/// <summary>
/// This example shows how to track the progress of a multipart upload
/// using the Amazon Simple Storage Service (Amazon S3) TransferUtility to
/// upload to an Amazon S3 bucket.
/// </summary>
public class TrackMPUUsingHighLevelAPI
{
    public static async Task Main()
    {
        string bucketName = "doc-example-bucket";
        string keyName = "sample_pic.png";
        string path = "filepath/directory/";
        string filePath = $"{path}{keyName}";

        // If the AWS Region defined for your default user is different
        // from the Region where your Amazon S3 bucket is located,
        // pass the Region name to the Amazon S3 client object's constructor.
        // For example: RegionEndpoint.USWest2 or RegionEndpoint.USEast2.
        IAmazonS3 client = new AmazonS3Client();

        await TrackMPUAsync(client, bucketName, filePath, keyName);
    }

    /// <summary>
    /// Starts an Amazon S3 multipart upload and assigns an event handler to
    /// track the progress of the upload.
    /// </summary>
    /// <param name="client">The initialized Amazon S3 client object used to
    /// perform the multipart upload.</param>
    /// <param name="bucketName">The name of the bucket to which to upload
    /// the file.</param>
    /// <param name="filePath">The path, including the file name of the
    /// file to be uploaded to the Amazon S3 bucket.</param>
    /// <param name="keyName">The file name to be used in the
    /// destination Amazon S3 bucket.</param>
    public static async Task TrackMPUAsync(
        IAmazonS3 client,
        string bucketName,
        string filePath,
        string keyName)
```

```
        {
            try
            {
                var fileTransferUtility = new TransferUtility(client);

                // Use TransferUtilityUploadRequest to configure options.
                // In this example we subscribe to an event.
                var uploadRequest =
                    new TransferUtilityUploadRequest
                    {
                        BucketName = bucketName,
                        FilePath = filePath,
                        Key = keyName,
                    };

                uploadRequest.UploadProgressEvent +=
                    new EventHandler<UploadProgressArgs>(
                        UploadRequest_UploadPartProgressEvent);

                await fileTransferUtility.UploadAsync(uploadRequest);
                Console.WriteLine("Upload completed");
            }
            catch (AmazonS3Exception ex)
            {
                Console.WriteLine($"Error:: {ex.Message}");
            }
        }

        /// <summary>
        /// Event handler to check the progress of the multipart upload.
        /// </summary>
        /// <param name="sender">The object that raised the event.</param>
        /// <param name="e">The object that contains multipart upload
        /// information.</param>
        public static void UploadRequest_UploadPartProgressEvent(object sender,
UploadProgressArgs e)
        {
            // Process event.
            Console.WriteLine($"{e.TransferredBytes}/{e.TotalBytes}");
        }
    }
```

暗号化したオブジェクトをアップロードします。

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Security.Cryptography;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

/// <summary>
/// Uses the Amazon Simple Storage Service (Amazon S3) low level API to
/// perform a multipart upload to an Amazon S3 bucket.
/// </summary>
public class SSECLowLevelMPUcopyObject
{
    public static async Task Main()
    {
        string existingBucketName = "doc-example-bucket";
        string sourceKeyName = "sample_file.txt";
        string targetKeyName = "sample_file_copy.txt";
        string filePath = $"sample\\{targetKeyName}";

        // If the AWS Region defined for your default user is different
        // from the Region where your Amazon S3 bucket is located,
        // pass the Region name to the Amazon S3 client object's constructor.
        // For example: RegionEndpoint.USEast1.
        IAmazonS3 client = new AmazonS3Client();

        // Create the encryption key.
        var base64Key = CreateEncryptionKey();

        await CreateSampleObjUsingClientEncryptionKeyAsync(
            client,
            existingBucketName,
            sourceKeyName,
            filePath,
            base64Key);
    }

    /// <summary>
    /// Creates the encryption key to use with the multipart upload.
    /// </summary>
    /// <returns>A string containing the base64-encoded key for encrypting
```

```
        /// the multipart upload.</returns>
        public static string CreateEncryptionKey()
        {
            Aes aesEncryption = Aes.Create();
            aesEncryption.KeySize = 256;
            aesEncryption.GenerateKey();
            string base64Key = Convert.ToBase64String(aesEncryption.Key);
            return base64Key;
        }

        /// <summary>
        /// Creates and uploads an object using a multipart upload.
        /// </summary>
        /// <param name="client">The initialized Amazon S3 object used to
        /// initialize and perform the multipart upload.</param>
        /// <param name="existingBucketName">The name of the bucket to which
        /// the object will be uploaded.</param>
        /// <param name="sourceKeyName">The source object name.</param>
        /// <param name="filePath">The location of the source object.</param>
        /// <param name="base64Key">The encryption key to use with the upload.</
param>
        public static async Task CreateSampleObjUsingClientEncryptionKeyAsync(
            IAmazonS3 client,
            string existingBucketName,
            string sourceKeyName,
            string filePath,
            string base64Key)
        {
            List<UploadPartResponse> uploadResponses = new
 List<UploadPartResponse>();

            InitiateMultipartUploadRequest initiateRequest = new
 InitiateMultipartUploadRequest
            {
                BucketName = existingBucketName,
                Key = sourceKeyName,
                ServerSideEncryptionCustomerMethod =
 ServerSideEncryptionCustomerMethod.AES256,
                ServerSideEncryptionCustomerProvidedKey = base64Key,
            };

            InitiateMultipartUploadResponse initResponse =
               await client.InitiateMultipartUploadAsync(initiateRequest);
```

```
            long contentLength = new FileInfo(filePath).Length;
            long partSize = 5 * (long)Math.Pow(2, 20); // 5 MB

            try
            {
                long filePosition = 0;
                for (int i = 1; filePosition < contentLength; i++)
                {
                    UploadPartRequest uploadRequest = new UploadPartRequest
                    {
                        BucketName = existingBucketName,
                        Key = sourceKeyName,
                        UploadId = initResponse.UploadId,
                        PartNumber = i,
                        PartSize = partSize,
                        FilePosition = filePosition,
                        FilePath = filePath,
                        ServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
                        ServerSideEncryptionCustomerProvidedKey = base64Key,
                    };

                    // Upload part and add response to our list.
                    uploadResponses.Add(await
client.UploadPartAsync(uploadRequest));

                    filePosition += partSize;
                }

                CompleteMultipartUploadRequest completeRequest = new
CompleteMultipartUploadRequest
                {
                    BucketName = existingBucketName,
                    Key = sourceKeyName,
                    UploadId = initResponse.UploadId,
                };
                completeRequest.AddPartETags(uploadResponses);

                CompleteMultipartUploadResponse completeUploadResponse =
                    await client.CompleteMultipartUploadAsync(completeRequest);
            }
            catch (Exception exception)
            {
                Console.WriteLine($"Exception occurred: {exception.Message}");
```

```
                // If there was an error, abort the multipart upload.
                AbortMultipartUploadRequest abortMPURequest = new
 AbortMultipartUploadRequest
                {
                    BucketName = existingBucketName,
                    Key = sourceKeyName,
                    UploadId = initResponse.UploadId,
                };

                await client.AbortMultipartUploadAsync(abortMPURequest);
            }
        }
    }
```

サーバーレスサンプル

Amazon S3 トリガーから Lambda 関数を呼び出す

次のコード例は、S3 バケットにオブジェクトをアップロードすることによってトリガーされるイベントを受け取る Lambda 関数を実装する方法を示しています。この関数は、イベントパラメータから S3 バケット名とオブジェクトキーを取得し、Amazon S3 API を呼び出してオブジェクトのコンテンツタイプを取得してログに記録します。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

.NET を使用して Lambda で S3 イベントを消費します。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Threading.Tasks;
using Amazon.Lambda.Core;
using Amazon.S3;
```

```
using System;
using Amazon.Lambda.S3Events;
using System.Web;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace S3Integration
{
    public class Function
    {
        private static AmazonS3Client _s3Client;
        public Function() : this(null)
        {
        }

        internal Function(AmazonS3Client s3Client)
        {
            _s3Client = s3Client ?? new AmazonS3Client();
        }

        public async Task<string> Handler(S3Event evt, ILambdaContext context)
        {
            try
            {
                if (evt.Records.Count <= 0)
                {
                    context.Logger.LogLine("Empty S3 Event received");
                    return string.Empty;
                }

                var bucket = evt.Records[0].S3.Bucket.Name;
                var key = HttpUtility.UrlDecode(evt.Records[0].S3.Object.Key);

                context.Logger.LogLine($"Request is for {bucket} and {key}");

                var objectResult = await _s3Client.GetObjectAsync(bucket, key);

                context.Logger.LogLine($"Returning {objectResult.Key}");

                return objectResult.Key;
            }
```

```
            catch (Exception e)
            {
                context.Logger.LogLine($"Error processing request - {e.Message}");

                return string.Empty;
            }
        }
    }
}
```

を使用した S3 Glacier の例 AWS SDK for .NET

次のコード例は、S3 Glacier AWS SDK for .NET で 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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。ここでは GitHub、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。

開始方法

Hello Amazon S3 Glacier

次のコード例は、Amazon S3 Glacier の使用を開始する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
using Amazon.Glacier;
using Amazon.Glacier.Model;

namespace GlacierActions;

public static class HelloGlacier
{
    static async Task Main()
    {
        var glacierService = new AmazonGlacierClient();

        Console.WriteLine("Hello Amazon Glacier!");
        Console.WriteLine("Let's list your Glacier vaults:");

        // You can use await and any of the async methods to get a response.
        // Let's get the vaults using a paginator.
        var glacierVaultPaginator = glacierService.Paginators.ListVaults(
            new ListVaultsRequest { AccountId = "-" });

        await foreach (var vault in glacierVaultPaginator.VaultList)
        {
            Console.WriteLine($"{vault.CreationDate}:{vault.VaultName}, ARN:
{vault.VaultARN}");
        }
    }
}
```

- API の詳細については、「 API リファレンスListVaults」の「」を参照してください。 AWS SDK for .NET

トピック

- アクション

アクション

`AddTagsToVault`

次の例は、AddTagsToVault を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Add tags to the items in an Amazon S3 Glacier vault.
    /// </summary>
    /// <param name="vaultName">The name of the vault to add tags to.</param>
    /// <param name="key">The name of the object to tag.</param>
    /// <param name="value">The tag value to add.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AddTagsToVaultAsync(string vaultName, string key, string
value)
    {
        var request = new AddTagsToVaultRequest
        {
            Tags = new Dictionary<string, string>
                {
                    { key, value },
                },
            AccountId = "-",
            VaultName = vaultName,
        };

        var response = await _glacierService.AddTagsToVaultAsync(request);
        return response.HttpStatusCode == HttpStatusCode.NoContent;
    }
```

- API の詳細については、「 API リファレンスAddTagsToVault」の「」を参照してください。AWS SDK for .NET

`CreateVault`

次の例は、`CreateVault` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
/// Create an Amazon S3 Glacier vault.
/// </summary>
/// <param name="vaultName">The name of the vault to create.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> CreateVaultAsync(string vaultName)
{
    var request = new CreateVaultRequest
    {
        // Setting the AccountId to "-" means that
        // the account associated with the current
        // account will be used.
        AccountId = "-",
        VaultName = vaultName,
    };

    var response = await _glacierService.CreateVaultAsync(request);

    Console.WriteLine($"Created {vaultName} at: {response.Location}");

    return response.HttpStatusCode == HttpStatusCode.Created;
}
```

- API の詳細については、「 API リファレンスCreateVault」の「」を参照してください。 AWS SDK for .NET

`DescribeVault`

次の例は、DescribeVault を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Describe an Amazon S3 Glacier vault.
    /// </summary>
    /// <param name="vaultName">The name of the vault to describe.</param>
    /// <returns>The Amazon Resource Name (ARN) of the vault.</returns>
    public async Task<string> DescribeVaultAsync(string vaultName)
    {
        var request = new DescribeVaultRequest
        {
            AccountId = "-",
            VaultName = vaultName,
        };

        var response = await _glacierService.DescribeVaultAsync(request);

        // Display the information about the vault.
        Console.WriteLine($"{response.VaultName}\tARN: {response.VaultARN}");
        Console.WriteLine($"Created on: {response.CreationDate}\tNumber of Archives:
{response.NumberOfArchives}\tSize (in bytes): {response.SizeInBytes}");
        if (response.LastInventoryDate != DateTime.MinValue)
        {
            Console.WriteLine($"Last inventory: {response.LastInventoryDate}");
        }

        return response.VaultARN;
    }
```

- API の詳細については、「 API リファレンスDescribeVault」の「」を参照してください。AWS SDK for .NET

`InitiateJob`

次の例は、InitiateJob を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ボールトからアーカイブを取得します。この例では、 ArchiveTransferManager クラスを使用します。API の詳細については、「」を参照してくださいArchiveTransferManager。

```
   /// <summary>
   /// Download an archive from an Amazon S3 Glacier vault using the Archive
   /// Transfer Manager.
   /// </summary>
   /// <param name="vaultName">The name of the vault containing the object.</param>
   /// <param name="archiveId">The Id of the archive to download.</param>
   /// <param name="localFilePath">The local directory where the file will
   /// be stored after download.</param>
   /// <returns>Async Task.</returns>
   public async Task<bool> DownloadArchiveWithArchiveManagerAsync(string vaultName,
string archiveId, string localFilePath)
   {
       try
       {
           var manager = new ArchiveTransferManager(_glacierService);

           var options = new DownloadOptions
           {
               StreamTransferProgress = Progress!,
           };

           // Download an archive.
           Console.WriteLine("Initiating the archive retrieval job and then polling
SQS queue for the archive to be available.");
           Console.WriteLine("When the archive is available, downloading will
begin.");
           await manager.DownloadAsync(vaultName, archiveId, localFilePath,
options);
```

```
            return true;
        }
        catch (AmazonGlacierException ex)
        {
            Console.WriteLine(ex.Message);
            return false;
        }
    }

    /// <summary>
    /// Event handler to track the progress of the Archive Transfer Manager.
    /// </summary>
    /// <param name="sender">The object that raised the event.</param>
    /// <param name="args">The argument values from the object that raised the
    /// event.</param>
    static void Progress(object sender, StreamTransferProgressArgs args)
    {
        if (args.PercentDone != _currentPercentage)
        {
            _currentPercentage = args.PercentDone;
            Console.WriteLine($"Downloaded {_currentPercentage}%");
        }
    }
```

- API の詳細については、「 API リファレンスInitiateJob」の「」を参照してください。 AWS SDK for .NET

`ListJobs`

次の例は、`ListJobs` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// List Amazon S3 Glacier jobs.
    /// </summary>
    /// <param name="vaultName">The name of the vault to list jobs for.</param>
    /// <returns>A list of Amazon S3 Glacier jobs.</returns>
    public async Task<List<GlacierJobDescription>> ListJobsAsync(string vaultName)
    {
        var request = new ListJobsRequest
        {
            // Using a hyphen "-" for the Account Id will
            // cause the SDK to use the Account Id associated
            // with the current account.
            AccountId = "-",
            VaultName = vaultName,
        };

        var response = await _glacierService.ListJobsAsync(request);

        return response.JobList;
    }
```

- API の詳細については、「 API リファレンスListJobs」の「」を参照してください。 AWS SDK for .NET

`ListTagsForVault`

次の例は、`ListTagsForVault` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// List tags for an Amazon S3 Glacier vault.
    /// </summary>
```

```
    /// <param name="vaultName">The name of the vault to list tags for.</param>
    /// <returns>A dictionary listing the tags attached to each object in the
    /// vault and its tags.</returns>
    public async Task<Dictionary<string, string>> ListTagsForVaultAsync(string
vaultName)
    {
        var request = new ListTagsForVaultRequest
        {
            // Using a hyphen "-" for the Account Id will
            // cause the SDK to use the Account Id associated
            // with the default user.
            AccountId = "-",
            VaultName = vaultName,
        };

        var response = await _glacierService.ListTagsForVaultAsync(request);

        return response.Tags;
    }
```

- API の詳細については、「 API リファレンスListTagsForVault」の「」を参照してください。 AWS SDK for .NET

`ListVaults`

次の例は、`ListVaults` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// List the Amazon S3 Glacier vaults associated with the current account.
    /// </summary>
    /// <returns>A list containing information about each vault.</returns>
```

```
    public async Task<List<DescribeVaultOutput>> ListVaultsAsync()
    {
        var glacierVaultPaginator = _glacierService.Paginators.ListVaults(
            new ListVaultsRequest { AccountId = "-" });
        var vaultList = new List<DescribeVaultOutput>();

        await foreach (var vault in glacierVaultPaginator.VaultList)
        {
            vaultList.Add(vault);
        }

        return vaultList;
    }
```

- API の詳細については、「 API リファレンスListVaults」の「」を参照してください。 AWS SDK for .NET

`UploadArchive`

次の例は、UploadArchive を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Upload an object to an Amazon S3 Glacier vault.
    /// </summary>
    /// <param name="vaultName">The name of the Amazon S3 Glacier vault to upload
    /// the archive to.</param>
    /// <param name="archiveFilePath">The file path of the archive to upload to the
vault.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<string> UploadArchiveWithArchiveManager(string vaultName,
string archiveFilePath)
```

```
    {
        try
        {
            var manager = new ArchiveTransferManager(_glacierService);

            // Upload an archive.
            var response = await manager.UploadAsync(vaultName, "upload archive
test", archiveFilePath);
            return response.ArchiveId;
        }
        catch (AmazonGlacierException ex)
        {
            Console.WriteLine(ex.Message);
            return string.Empty;
        }
    }
```

- API の詳細については、「 API リファレンスUploadArchive」の「」を参照してください。 AWS SDK for .NET

SageMaker を使用した の例 AWS SDK for .NET

次のコード例は、 AWS SDK for .NET で 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 SageMaker。

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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。ここでは GitHub、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。

開始方法

こんにち SageMakerは

次のコード例は、 の使用を開始する方法を示しています SageMaker。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
using Amazon.SageMaker;
using Amazon.SageMaker.Model;

namespace SageMakerActions;

public static class HelloSageMaker
{
    static async Task Main(string[] args)
    {
        var sageMakerClient = new AmazonSageMakerClient();

        Console.WriteLine($"Hello Amazon SageMaker! Let's list some of your notebook
 instances:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get the first five notebook instances.
        var response = await sageMakerClient.ListNotebookInstancesAsync(
            new ListNotebookInstancesRequest()
            {
                MaxResults = 5
            });

        if (!response.NotebookInstances.Any())
        {
            Console.WriteLine($"No notebook instances found.");
            Console.WriteLine("See https://docs.aws.amazon.com/sagemaker/latest/dg/
howitworks-create-ws.html to create one.");
        }

        foreach (var notebookInstance in response.NotebookInstances)
        {
```

```
            Console.WriteLine($"\tInstance:
 {notebookInstance.NotebookInstanceName}");
            Console.WriteLine($"\tArn: {notebookInstance.NotebookInstanceArn}");
            Console.WriteLine($"\tCreation Date:
 {notebookInstance.CreationTime.ToShortDateString()}");
            Console.WriteLine();
        }
    }
}
```

- API の詳細については、「 API リファレンスListNotebookInstances」の「」を参照してください。 AWS SDK for .NET

トピック

- アクション
- シナリオ

アクション

`CreatePipeline`

次の例は、CreatePipeline を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Create a pipeline from a JSON definition, or update it if the pipeline
 already exists.
    /// </summary>
    /// <returns>The Amazon Resource Name (ARN) of the pipeline.</returns>
    public async Task<string> SetupPipeline(string pipelineJson, string roleArn,
 string name, string description, string displayName)
```

```
    {
        try
        {
            var updateResponse = await _amazonSageMaker.UpdatePipelineAsync(
                new UpdatePipelineRequest()
                {
                    PipelineDefinition = pipelineJson,
                    PipelineDescription = description,
                    PipelineDisplayName = displayName,
                    PipelineName = name,
                    RoleArn = roleArn
                });
            return updateResponse.PipelineArn;
        }
        catch (Amazon.SageMaker.Model.ResourceNotFoundException)
        {
            var createResponse = await _amazonSageMaker.CreatePipelineAsync(
                new CreatePipelineRequest()
                {
                    PipelineDefinition = pipelineJson,
                    PipelineDescription = description,
                    PipelineDisplayName = displayName,
                    PipelineName = name,
                    RoleArn = roleArn
                });

            return createResponse.PipelineArn;
        }
    }
```

- API の詳細については、「 API リファレンスCreatePipeline」の「」を参照してください。AWS SDK for .NET

DeletePipeline

次の例は、DeletePipeline を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
/// Delete a SageMaker pipeline by name.
/// </summary>
/// <param name="pipelineName">The name of the pipeline to delete.</param>
/// <returns>The ARN of the pipeline.</returns>
public async Task<string> DeletePipelineByName(string pipelineName)
{
    var deleteResponse = await _amazonSageMaker.DeletePipelineAsync(
        new DeletePipelineRequest()
        {
            PipelineName = pipelineName
        });

    return deleteResponse.PipelineArn;
}
```

- API の詳細については、「 API リファレンスDeletePipeline」の「」を参照してください。AWS SDK for .NET

`DescribePipelineExecution`

次の例は、`DescribePipelineExecution` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Check the status of a run.
    /// </summary>
    /// <param name="pipelineExecutionArn">The ARN.</param>
    /// <returns>The status of the pipeline.</returns>
    public async Task<PipelineExecutionStatus> CheckPipelineExecutionStatus(string
pipelineExecutionArn)
    {
        var describeResponse = await
_amazonSageMaker.DescribePipelineExecutionAsync(
            new DescribePipelineExecutionRequest()
            {
                PipelineExecutionArn = pipelineExecutionArn
            });

        return describeResponse.PipelineExecutionStatus;
    }
```

- API の詳細については、「 API リファレンスDescribePipelineExecution」の「」を参照してください。 AWS SDK for .NET

`StartPipelineExecution`

次の例は、`StartPipelineExecution` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Run a pipeline with input and output file locations.
    /// </summary>
    /// <param name="queueUrl">The URL for the queue to use for pipeline
callbacks.</param>
    /// <param name="inputLocationUrl">The input location in Amazon Simple Storage
Service (Amazon S3).</param>
```

```
    /// <param name="outputLocationUrl">The output location in Amazon S3.</param>
    /// <param name="pipelineName">The name of the pipeline.</param>
    /// <param name="executionRoleArn">The ARN of the role.</param>
    /// <returns>The ARN of the pipeline run.</returns>
    public async Task<string> ExecutePipeline(
        string queueUrl,
        string inputLocationUrl,
        string outputLocationUrl,
        string pipelineName,
        string executionRoleArn)
    {
        var inputConfig = new VectorEnrichmentJobInputConfig()
        {
            DataSourceConfig = new()
            {
                S3Data = new VectorEnrichmentJobS3Data()
                {
                    S3Uri = inputLocationUrl
                }
            },
            DocumentType = VectorEnrichmentJobDocumentType.CSV
        };

        var exportConfig = new ExportVectorEnrichmentJobOutputConfig()
        {
            S3Data = new VectorEnrichmentJobS3Data()
            {
                S3Uri = outputLocationUrl
            }
        };

        var jobConfig = new VectorEnrichmentJobConfig()
        {
            ReverseGeocodingConfig = new ReverseGeocodingConfig()
            {
                XAttributeName = "Longitude",
                YAttributeName = "Latitude"
            }
        };

#pragma warning disable SageMaker1002 // Property value does not match required
 pattern is allowed here to match the pipeline definition.
        var startExecutionResponse = await
 _amazonSageMaker.StartPipelineExecutionAsync(
```

```
            new StartPipelineExecutionRequest()
            {
                PipelineName = pipelineName,
                PipelineExecutionDisplayName = pipelineName + "-example-execution",
                PipelineParameters = new List<Parameter>()
                {
                    new Parameter() { Name = "parameter_execution_role", Value =
 executionRoleArn },
                    new Parameter() { Name = "parameter_queue_url", Value =
 queueUrl },
                    new Parameter() { Name = "parameter_vej_input_config", Value =
 JsonSerializer.Serialize(inputConfig) },
                    new Parameter() { Name = "parameter_vej_export_config", Value =
 JsonSerializer.Serialize(exportConfig) },
                    new Parameter() { Name = "parameter_step_1_vej_config", Value =
 JsonSerializer.Serialize(jobConfig) }
                }
            });
#pragma warning restore SageMaker1002
        return startExecutionResponse.PipelineExecutionArn;
    }
```

- API の詳細については、「 API リファレンスStartPipelineExecution」の「」を参照してください。 AWS SDK for .NET

`UpdatePipeline`

次の例は、UpdatePipeline を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Create a pipeline from a JSON definition, or update it if the pipeline
 already exists.
```

```
    /// </summary>
    /// <returns>The Amazon Resource Name (ARN) of the pipeline.</returns>
    public async Task<string> SetupPipeline(string pipelineJson, string roleArn,
string name, string description, string displayName)
    {
        try
        {
            var updateResponse = await _amazonSageMaker.UpdatePipelineAsync(
                new UpdatePipelineRequest()
                {
                    PipelineDefinition = pipelineJson,
                    PipelineDescription = description,
                    PipelineDisplayName = displayName,
                    PipelineName = name,
                    RoleArn = roleArn
                });
            return updateResponse.PipelineArn;
        }
        catch (Amazon.SageMaker.Model.ResourceNotFoundException)
        {
            var createResponse = await _amazonSageMaker.CreatePipelineAsync(
                new CreatePipelineRequest()
                {
                    PipelineDefinition = pipelineJson,
                    PipelineDescription = description,
                    PipelineDisplayName = displayName,
                    PipelineName = name,
                    RoleArn = roleArn
                });

            return createResponse.PipelineArn;
        }
    }
```

- API の詳細については、「 API リファレンスUpdatePipeline」の「」を参照してください。AWS SDK for .NET

シナリオ

地理空間ジョブとパイプラインの使用を開始する

次のコードサンプルは、以下の操作方法を示しています。

- パイプラインのリソースを設定します。
- 地理空間ジョブを実行するパイプラインを設定します。
- パイプラインの実行を開始します。
- ジョブ実行のステータスをモニタリングします。
- パイプラインの出力を表示します。
- リソースをクリーンアップします。

詳細については、「Community.aws で AWS SDKs を使用して SageMaker パイプラインを作成および実行する」を参照してください。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

SageMaker オペレーションをラップするクラスを作成します。

```
using System.Text.Json;
using Amazon.SageMaker;
using Amazon.SageMaker.Model;
using Amazon.SageMakerGeospatial;
using Amazon.SageMakerGeospatial.Model;

namespace SageMakerActions;

/// <summary>
/// Wrapper class for Amazon SageMaker actions and logic.
/// </summary>
public class SageMakerWrapper
{
    private readonly IAmazonSageMaker _amazonSageMaker;
    public SageMakerWrapper(IAmazonSageMaker amazonSageMaker)
    {
        _amazonSageMaker = amazonSageMaker;
    }
```

```
   /// <summary>
   /// Create a pipeline from a JSON definition, or update it if the pipeline
already exists.
   /// </summary>
   /// <returns>The Amazon Resource Name (ARN) of the pipeline.</returns>
   public async Task<string> SetupPipeline(string pipelineJson, string roleArn,
string name, string description, string displayName)
   {
       try
       {
           var updateResponse = await _amazonSageMaker.UpdatePipelineAsync(
               new UpdatePipelineRequest()
               {
                   PipelineDefinition = pipelineJson,
                   PipelineDescription = description,
                   PipelineDisplayName = displayName,
                   PipelineName = name,
                   RoleArn = roleArn
               });
           return updateResponse.PipelineArn;
       }
       catch (Amazon.SageMaker.Model.ResourceNotFoundException)
       {
           var createResponse = await _amazonSageMaker.CreatePipelineAsync(
               new CreatePipelineRequest()
               {
                   PipelineDefinition = pipelineJson,
                   PipelineDescription = description,
                   PipelineDisplayName = displayName,
                   PipelineName = name,
                   RoleArn = roleArn
               });

           return createResponse.PipelineArn;
       }
   }

   /// <summary>
   /// Run a pipeline with input and output file locations.
   /// </summary>
   /// <param name="queueUrl">The URL for the queue to use for pipeline
callbacks.</param>
```

```
    /// <param name="inputLocationUrl">The input location in Amazon Simple Storage
 Service (Amazon S3).</param>
    /// <param name="outputLocationUrl">The output location in Amazon S3.</param>
    /// <param name="pipelineName">The name of the pipeline.</param>
    /// <param name="executionRoleArn">The ARN of the role.</param>
    /// <returns>The ARN of the pipeline run.</returns>
    public async Task<string> ExecutePipeline(
        string queueUrl,
        string inputLocationUrl,
        string outputLocationUrl,
        string pipelineName,
        string executionRoleArn)
    {
        var inputConfig = new VectorEnrichmentJobInputConfig()
        {
            DataSourceConfig = new()
            {
                S3Data = new VectorEnrichmentJobS3Data()
                {
                    S3Uri = inputLocationUrl
                }
            },
            DocumentType = VectorEnrichmentJobDocumentType.CSV
        };

        var exportConfig = new ExportVectorEnrichmentJobOutputConfig()
        {
            S3Data = new VectorEnrichmentJobS3Data()
            {
                S3Uri = outputLocationUrl
            }
        };

        var jobConfig = new VectorEnrichmentJobConfig()
        {
            ReverseGeocodingConfig = new ReverseGeocodingConfig()
            {
                XAttributeName = "Longitude",
                YAttributeName = "Latitude"
            }
        };

#pragma warning disable SageMaker1002 // Property value does not match required
 pattern is allowed here to match the pipeline definition.
```

```
        var startExecutionResponse = await
 _amazonSageMaker.StartPipelineExecutionAsync(
            new StartPipelineExecutionRequest()
            {
                PipelineName = pipelineName,
                PipelineExecutionDisplayName = pipelineName + "-example-execution",
                PipelineParameters = new List<Parameter>()
                {
                    new Parameter() { Name = "parameter_execution_role", Value =
 executionRoleArn },
                    new Parameter() { Name = "parameter_queue_url", Value =
 queueUrl },
                    new Parameter() { Name = "parameter_vej_input_config", Value =
 JsonSerializer.Serialize(inputConfig) },
                    new Parameter() { Name = "parameter_vej_export_config", Value =
 JsonSerializer.Serialize(exportConfig) },
                    new Parameter() { Name = "parameter_step_1_vej_config", Value =
 JsonSerializer.Serialize(jobConfig) }
                }
            });
#pragma warning restore SageMaker1002
        return startExecutionResponse.PipelineExecutionArn;
    }

    /// <summary>
    /// Check the status of a run.
    /// </summary>
    /// <param name="pipelineExecutionArn">The ARN.</param>
    /// <returns>The status of the pipeline.</returns>
    public async Task<PipelineExecutionStatus> CheckPipelineExecutionStatus(string
 pipelineExecutionArn)
    {
        var describeResponse = await
 _amazonSageMaker.DescribePipelineExecutionAsync(
            new DescribePipelineExecutionRequest()
            {
                PipelineExecutionArn = pipelineExecutionArn
            });

        return describeResponse.PipelineExecutionStatus;
    }

    /// <summary>
    /// Delete a SageMaker pipeline by name.
```

```
    /// </summary>
    /// <param name="pipelineName">The name of the pipeline to delete.</param>
    /// <returns>The ARN of the pipeline.</returns>
    public async Task<string> DeletePipelineByName(string pipelineName)
    {
        var deleteResponse = await _amazonSageMaker.DeletePipelineAsync(
            new DeletePipelineRequest()
            {
                PipelineName = pipelineName
            });

        return deleteResponse.PipelineArn;
    }
}
```

SageMaker パイプラインからのコールバックを処理する関数を作成します。

```
using System.Text.Json;
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;
using Amazon.SageMaker;
using Amazon.SageMaker.Model;
using Amazon.SageMakerGeospatial;
using Amazon.SageMakerGeospatial.Model;

// Assembly attribute to enable the AWS Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace SageMakerLambda;

/// <summary>
/// The AWS Lambda function handler for the Amazon SageMaker pipeline.
/// </summary>
public class SageMakerLambdaFunction
{
    /// <summary>
    /// Default constructor. This constructor is used by AWS Lambda to construct the
 instance. When invoked in a Lambda environment
```

```
   /// the AWS credentials will come from the AWS Identity and Access Management
(IAM) role associated with the function. The AWS Region will be set to the
   /// Region that the Lambda function is running in.
   /// </summary>
   public SageMakerLambdaFunction()
   {
   }

   /// <summary>
   /// The AWS Lambda function handler that processes events from the SageMaker
pipeline and starts a job or export.
   /// </summary>
   /// <param name="request">The custom SageMaker pipeline request object.</param>
   /// <param name="context">The Lambda context.</param>
   /// <returns>The dictionary of output parameters.</returns>
   public async Task<Dictionary<string, string>> FunctionHandler(PipelineRequest
request, ILambdaContext context)
   {
       var geoSpatialClient = new AmazonSageMakerGeospatialClient();
       var sageMakerClient = new AmazonSageMakerClient();
       var responseDictionary = new Dictionary<string, string>();
       context.Logger.LogInformation("Function handler started with request: " +
JsonSerializer.Serialize(request));
       if (request.Records != null && request.Records.Any())
       {
           context.Logger.LogInformation("Records found, this is a queue event.
Processing the queue records.");
           foreach (var message in request.Records)
           {
               await ProcessMessageAsync(message, context, geoSpatialClient,
sageMakerClient);
           }
       }
       else if (!string.IsNullOrEmpty(request.vej_export_config))
       {
           context.Logger.LogInformation("Export configuration found, this is an
export. Start the Vector Enrichment Job (VEJ) export.");

           var outputConfig =
               JsonSerializer.Deserialize<ExportVectorEnrichmentJobOutputConfig>(
                   request.vej_export_config);

           var exportResponse = await
geoSpatialClient.ExportVectorEnrichmentJobAsync(
```

```
                new ExportVectorEnrichmentJobRequest()
                {
                    Arn = request.vej_arn,
                    ExecutionRoleArn = request.Role,
                    OutputConfig = outputConfig
                });
            context.Logger.LogInformation($"Export response:
{JsonSerializer.Serialize(exportResponse)}");
            responseDictionary = new Dictionary<string, string>
            {
                { "export_eoj_status", exportResponse.ExportStatus.ToString() },
                { "vej_arn", exportResponse.Arn }
            };
        }
        else if (!string.IsNullOrEmpty(request.vej_name))
        {
            context.Logger.LogInformation("Vector Enrichment Job name found,
starting the job.");
            var inputConfig =
                JsonSerializer.Deserialize<VectorEnrichmentJobInputConfig>(
                    request.vej_input_config);

            var jobConfig =
                JsonSerializer.Deserialize<VectorEnrichmentJobConfig>(
                    request.vej_config);

            var jobResponse = await geoSpatialClient.StartVectorEnrichmentJobAsync(
                new StartVectorEnrichmentJobRequest()
                {
                    ExecutionRoleArn = request.Role,
                    InputConfig = inputConfig,
                    Name = request.vej_name,
                    JobConfig = jobConfig

                });
            context.Logger.LogInformation("Job response: " +
JsonSerializer.Serialize(jobResponse));
            responseDictionary = new Dictionary<string, string>
            {
                { "vej_arn", jobResponse.Arn },
                { "statusCode", jobResponse.HttpStatusCode.ToString() }
            };
        }
        return responseDictionary;
```

```
    }

    /// <summary>
    /// Process a queue message and check the status of a SageMaker job.
    /// </summary>
    /// <param name="message">The queue message.</param>
    /// <param name="context">The Lambda context.</param>
    /// <param name="geoClient">The SageMaker GeoSpatial client.</param>
    /// <param name="sageMakerClient">The SageMaker client.</param>
    /// <returns>Async task.</returns>
    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
ILambdaContext context,
        AmazonSageMakerGeospatialClient geoClient, AmazonSageMakerClient
sageMakerClient)
    {
        context.Logger.LogInformation($"Processed message {message.Body}");

        // Get information about the SageMaker job.
        var payload = JsonSerializer.Deserialize<QueuePayload>(message.Body);
        context.Logger.LogInformation($"Payload token {payload!.token}");
        var token = payload.token;

        if (payload.arguments.ContainsKey("vej_arn"))
        {
            // Use the job ARN and the token to get the job status.
            var job_arn = payload.arguments["vej_arn"];
            context.Logger.LogInformation($"Token: {token}, arn {job_arn}");

            var jobInfo = geoClient.GetVectorEnrichmentJobAsync(
                new GetVectorEnrichmentJobRequest()
                {
                    Arn = job_arn
                });
            context.Logger.LogInformation("Job info: " +
JsonSerializer.Serialize(jobInfo));
            if (jobInfo.Result.Status == VectorEnrichmentJobStatus.COMPLETED)
            {
                context.Logger.LogInformation($"Status completed, resuming
pipeline...");
                await sageMakerClient.SendPipelineExecutionStepSuccessAsync(
                    new SendPipelineExecutionStepSuccessRequest()
                    {
                        CallbackToken = token,
                        OutputParameters = new List<OutputParameter>()
```

```
                        {
                            new OutputParameter()
                                { Name = "export_status", Value =
 jobInfo.Result.Status }
                        }
                    });
            }
            else if (jobInfo.Result.Status == VectorEnrichmentJobStatus.FAILED)
            {
                context.Logger.LogInformation($"Status failed, stopping
 pipeline...");
                await sageMakerClient.SendPipelineExecutionStepFailureAsync(
                    new SendPipelineExecutionStepFailureRequest()
                    {
                        CallbackToken = token,
                        FailureReason = jobInfo.Result.ErrorDetails.ErrorMessage
                    });
            }
            else if (jobInfo.Result.Status == VectorEnrichmentJobStatus.IN_PROGRESS)
            {
                // Put this message back in the queue to reprocess later.
                context.Logger.LogInformation(
                    $"Status still in progress, check back later.");
                throw new("Job still running.");
            }
        }
    }
}
```

コマンドプロンプトからインタラクティブのシナリオを実行します。

```
public static class PipelineWorkflow
{
    public static IAmazonIdentityManagementService _iamClient = null!;
    public static SageMakerWrapper _sageMakerWrapper = null!;
    public static IAmazonSQS _sqsClient = null!;
    public static IAmazonS3 _s3Client = null!;
    public static IAmazonLambda _lambdaClient = null!;
    public static IConfiguration _configuration = null!;

    public static string lambdaFunctionName = "SageMakerExampleFunction";
    public static string sageMakerRoleName = "SageMakerExampleRole";
```

```
    public static string lambdaRoleName = "SageMakerExampleLambdaRole";

    private static string[] lambdaRolePolicies = null!;
    private static string[] sageMakerRolePolicies = null!;

    static async Task Main(string[] args)
    {
        var options = new AWSOptions() { Region = RegionEndpoint.USWest2 };
        // Set up dependency injection for the AWS 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<IAmazonIdentityManagementService>(options)
                    .AddAWSService<IAmazonEC2>(options)
                    .AddAWSService<IAmazonSageMaker>(options)
                    .AddAWSService<IAmazonSageMakerGeospatial>(options)
                    .AddAWSService<IAmazonSQS>(options)
                    .AddAWSService<IAmazonS3>(options)
                    .AddAWSService<IAmazonLambda>(options)
                    .AddTransient<SageMakerWrapper>()
        )
        .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();

        ServicesSetup(host);
        string queueUrl = "";
        string queueName = _configuration["queueName"];
        string bucketName = _configuration["bucketName"];
        var pipelineName = _configuration["pipelineName"];

        try
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine(
```

```
                "Welcome to the Amazon SageMaker pipeline example scenario.");
            Console.WriteLine(
                "\nThis example workflow will guide you through setting up and
 running an" +
                "\nAmazon SageMaker pipeline. The pipeline uses an AWS Lambda
 function and an" +
                "\nAmazon SQS Queue. It runs a vector enrichment reverse geocode job
 to" +
                "\nreverse geocode addresses in an input file and store the results
 in an export file.");
            Console.WriteLine(new string('-', 80));

            Console.WriteLine(new string('-', 80));
            Console.WriteLine(
                "First, we will set up the roles, functions, and queue needed by the
 SageMaker pipeline.");
            Console.WriteLine(new string('-', 80));

            var lambdaRoleArn = await CreateLambdaRole();
            var sageMakerRoleArn = await CreateSageMakerRole();
            var functionArn = await SetupLambda(lambdaRoleArn, true);
            queueUrl = await SetupQueue(queueName);
            await SetupBucket(bucketName);

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Now we can create and run our pipeline.");
            Console.WriteLine(new string('-', 80));

            await SetupPipeline(sageMakerRoleArn, functionArn, pipelineName);
            var executionArn = await ExecutePipeline(queueUrl, sageMakerRoleArn,
 pipelineName, bucketName);
            await WaitForPipelineExecution(executionArn);

            await GetOutputResults(bucketName);

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("The pipeline has completed. To view the pipeline and
 runs " +
                              "in SageMaker Studio, follow these instructions:" +
                              "\nhttps://docs.aws.amazon.com/sagemaker/latest/dg/
pipelines-studio.html");
            Console.WriteLine(new string('-', 80));

            Console.WriteLine(new string('-', 80));
```

```
            Console.WriteLine("Finally, let's clean up our resources.");
            Console.WriteLine(new string('-', 80));

            await CleanupResources(true, queueUrl, pipelineName, bucketName);

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("SageMaker pipeline scenario is complete.");
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
            await CleanupResources(true, queueUrl, pipelineName, bucketName);
            Console.WriteLine(new string('-', 80));
        }
    }

    /// <summary>
    /// Populate the services for use within the console application.
    /// </summary>
    /// <param name="host">The services host.</param>
    private static void ServicesSetup(IHost host)
    {
        _sageMakerWrapper = host.Services.GetRequiredService<SageMakerWrapper>();
        _iamClient =
host.Services.GetRequiredService<IAmazonIdentityManagementService>();
        _sqsClient = host.Services.GetRequiredService<IAmazonSQS>();
        _s3Client = host.Services.GetRequiredService<IAmazonS3>();
        _lambdaClient = host.Services.GetRequiredService<IAmazonLambda>();
    }

    /// <summary>
    /// Set up AWS Lambda, either by updating an existing function or creating a new
function.
    /// </summary>
    /// <param name="roleArn">The role Amazon Resource Name (ARN) to use for the
Lambda function.</param>
    /// <param name="askUser">True to ask the user before updating.</param>
    /// <returns>The ARN of the function.</returns>
    public static async Task<string> SetupLambda(string roleArn, bool askUser)
    {
        Console.WriteLine(new string('-', 80));
```

```
        Console.WriteLine("Setting up the Lambda function for the pipeline.");
        var handlerName =
"SageMakerLambda::SageMakerLambda.SageMakerLambdaFunction::FunctionHandler";
        var functionArn = "";
        try
        {
            var functionInfo = await _lambdaClient.GetFunctionAsync(new
GetFunctionRequest()
            {
                FunctionName = lambdaFunctionName
            });

            var updateFunction = true;
            if (askUser)
            {
                updateFunction = GetYesNoResponse(
                    $"\tThe Lambda function {lambdaFunctionName} already exists, do
you want to update it?");
            }

            if (updateFunction)
            {
                // Update the Lambda function.
                using var zipMemoryStream = new MemoryStream(await
File.ReadAllBytesAsync("SageMakerLambda.zip"));
                await _lambdaClient.UpdateFunctionCodeAsync(
                    new UpdateFunctionCodeRequest()
                    {
                        FunctionName = lambdaFunctionName,
                        ZipFile = zipMemoryStream,
                    });
            }

            functionArn = functionInfo.Configuration.FunctionArn;
        }
        catch (ResourceNotFoundException)
        {
            Console.WriteLine($"\tThe Lambda function {lambdaFunctionName} was not
found, creating the new function.");

            // Create the function if it does not already exist.
            using var zipMemoryStream = new MemoryStream(await
File.ReadAllBytesAsync("SageMakerLambda.zip"));
            var createResult = await _lambdaClient.CreateFunctionAsync(
```

```
                new CreateFunctionRequest()
                {
                    FunctionName = lambdaFunctionName,
                    Runtime = Runtime.Dotnet6,
                    Description = "SageMaker example function.",
                    Code = new FunctionCode()
                    {
                        ZipFile = zipMemoryStream
                    },
                    Handler = handlerName,
                    Role = roleArn,
                    Timeout = 30
                });

            functionArn = createResult.FunctionArn;
        }

        Console.WriteLine($"\tLambda ready with ARN {functionArn}.");
        Console.WriteLine(new string('-', 80));
        return functionArn;
    }

    /// <summary>
    /// Create a role to be used by AWS Lambda. Does not create the role if it
already exists.
    /// </summary>
    /// <returns>The role ARN.</returns>
    public static async Task<string> CreateLambdaRole()
    {
        Console.WriteLine(new string('-', 80));

        lambdaRolePolicies = new string[]{
            "arn:aws:iam::aws:policy/AmazonSageMakerFullAccess",
            "arn:aws:iam::aws:policy/AmazonSQSFullAccess",
            "arn:aws:iam::aws:policy/service-role/" +
"AmazonSageMakerGeospatialFullAccess",
            "arn:aws:iam::aws:policy/service-role/" +
"AmazonSageMakerServiceCatalogProductsLambdaServiceRolePolicy",
            "arn:aws:iam::aws:policy/service-role/" +
"AWSLambdaSQSQueueExecutionRole"
        };

        var roleArn = await GetRoleArnIfExists(lambdaRoleName);
        if (!string.IsNullOrEmpty(roleArn))
```

```
        {
            return roleArn;
        }

        Console.WriteLine("\tCreating a role to for AWS Lambda to use.");

        var assumeRolePolicy = "{" +
                               "\"Version\": \"2012-10-17\"," +
                               "\"Statement\": [{" +
                                    "\"Effect\": \"Allow\"," +
                                    "\"Principal\": {" +
                                        $"\"Service\": [" +
                                            "\"sagemaker.amazonaws.com\"," +
                                            "\"sagemaker-geospatial.amazonaws.com
\"," +
                                            "\"lambda.amazonaws.com\"," +
                                            "\"s3.amazonaws.com\"" +
                                        "]" +
                                    "}," +
                                    "\"Action\": \"sts:AssumeRole\"" +
                               "}]" +
                           "}";

        var roleResult = await _iamClient!.CreateRoleAsync(
            new CreateRoleRequest()
            {
                AssumeRolePolicyDocument = assumeRolePolicy,
                Path = "/",
                RoleName = lambdaRoleName
            });
        foreach (var policy in lambdaRolePolicies)
        {
            await _iamClient.AttachRolePolicyAsync(
                new AttachRolePolicyRequest()
                {
                    PolicyArn = policy,
                    RoleName = lambdaRoleName
                });
        }

        // Allow time for the role to be ready.
        Thread.Sleep(10000);
        Console.WriteLine($"\tRole ready with ARN {roleResult.Role.Arn}.");
        Console.WriteLine(new string('-', 80));
```

```
        return roleResult.Role.Arn;
    }


    /// <summary>
    /// Create a role to be used by SageMaker.
    /// </summary>
    /// <returns>The role Amazon Resource Name (ARN).</returns>
    public static async Task<string> CreateSageMakerRole()
    {
        Console.WriteLine(new string('-', 80));

        sageMakerRolePolicies = new string[]{
            "arn:aws:iam::aws:policy/AmazonSageMakerFullAccess",
            "arn:aws:iam::aws:policy/AmazonSageMakerGeospatialFullAccess",
        };

        var roleArn = await GetRoleArnIfExists(sageMakerRoleName);
        if (!string.IsNullOrEmpty(roleArn))
        {
            return roleArn;
        }

        Console.WriteLine("\tCreating a role to use with SageMaker.");

        var assumeRolePolicy = "{" +
                                       "\"Version\": \"2012-10-17\"," +
                                       "\"Statement\": [{" +
                                           "\"Effect\": \"Allow\"," +
                                           "\"Principal\": {" +
                                               $"\"Service\": [" +
                                                   "\"sagemaker.amazonaws.com\"," +
                                                   "\"sagemaker-
geospatial.amazonaws.com\"," +
                                                   "\"lambda.amazonaws.com\"," +
                                                   "\"s3.amazonaws.com\"" +
                                               "]" +
                                           "}," +
                                           "\"Action\": \"sts:AssumeRole\"" +
                                       "}]" +
                                   "}";

        var roleResult = await _iamClient!.CreateRoleAsync(
```

```
            new CreateRoleRequest()
            {
                AssumeRolePolicyDocument = assumeRolePolicy,
                Path = "/",
                RoleName = sageMakerRoleName
            });

        foreach (var policy in sageMakerRolePolicies)
        {
            await _iamClient.AttachRolePolicyAsync(
                new AttachRolePolicyRequest()
                {
                    PolicyArn = policy,
                    RoleName = sageMakerRoleName
                });
        }

        // Allow time for the role to be ready.
        Thread.Sleep(10000);
        Console.WriteLine($"\tRole ready with ARN {roleResult.Role.Arn}.");
        Console.WriteLine(new string('-', 80));
        return roleResult.Role.Arn;
    }

    /// <summary>
    /// Set up the SQS queue to use with the pipeline.
    /// </summary>
    /// <param name="queueName">The name for the queue.</param>
    /// <returns>The URL for the queue.</returns>
    public static async Task<string> SetupQueue(string queueName)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Setting up queue {queueName}.");

        try
        {
            var queueInfo = await _sqsClient.GetQueueUrlAsync(new
GetQueueUrlRequest()
            { QueueName = queueName });
            return queueInfo.QueueUrl;
        }
        catch (QueueDoesNotExistException)
        {
            var attrs = new Dictionary<string, string>
```

```
            {
                {
                    QueueAttributeName.DelaySeconds,
                    "5"
                },
                {
                    QueueAttributeName.ReceiveMessageWaitTimeSeconds,
                    "5"
                },
                {
                    QueueAttributeName.VisibilityTimeout,
                    "300"
                },
            };

            var request = new CreateQueueRequest
            {
                Attributes = attrs,
                QueueName = queueName,
            };

            var response = await _sqsClient.CreateQueueAsync(request);
            Thread.Sleep(10000);
            await ConnectLambda(response.QueueUrl);
            Console.WriteLine($"\tQueue ready with Url {response.QueueUrl}.");
            Console.WriteLine(new string('-', 80));
            return response.QueueUrl;
        }
    }

    /// <summary>
    /// Connect the queue to the Lambda function as an event source.
    /// </summary>
    /// <param name="queueUrl">The URL for the queue.</param>
    /// <returns>Async task.</returns>
    public static async Task ConnectLambda(string queueUrl)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Connecting the Lambda function and queue for the
pipeline.");

        var queueAttributes = await _sqsClient.GetQueueAttributesAsync(
            new GetQueueAttributesRequest() { QueueUrl = queueUrl, AttributeNames =
new List<string>() { "All" } });
```

```
        var queueArn = queueAttributes.QueueARN;

        var eventSource = await _lambdaClient.ListEventSourceMappingsAsync(
             new ListEventSourceMappingsRequest()
             {
                 FunctionName = lambdaFunctionName
             });

        if (!eventSource.EventSourceMappings.Any())
        {
            // Only add the event source mapping if it does not already exist.
            await _lambdaClient.CreateEventSourceMappingAsync(
                new CreateEventSourceMappingRequest()
                {
                    EventSourceArn = queueArn,
                    FunctionName = lambdaFunctionName,
                    Enabled = true
                });
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Set up the bucket to use for pipeline input and output.
    /// </summary>
    /// <param name="bucketName">The name for the bucket.</param>
    /// <returns>Async task.</returns>
    public static async Task SetupBucket(string bucketName)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Setting up bucket {bucketName}.");

        var bucketExists = await
Amazon.S3.Util.AmazonS3Util.DoesS3BucketExistV2Async(_s3Client,
            bucketName);

        if (!bucketExists)
        {
            await _s3Client.PutBucketAsync(new PutBucketRequest()
            {
                BucketName = bucketName,
                BucketRegion = S3Region.USWest2
            });
```

```
            Thread.Sleep(5000);

            await _s3Client.PutObjectAsync(new PutObjectRequest()
            {
                BucketName = bucketName,
                Key = "samplefiles/latlongtest.csv",
                FilePath = "latlongtest.csv"
            });
        }

        Console.WriteLine($"\tBucket {bucketName} ready.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Display some results from the output directory.
    /// </summary>
    /// <param name="bucketName">The name for the bucket.</param>
    /// <returns>Async task.</returns>
    public static async Task<string> GetOutputResults(string bucketName)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Getting output results {bucketName}.");
        string outputKey = "";
        Thread.Sleep(15000);
        var outputFiles = await _s3Client.ListObjectsAsync(
              new ListObjectsRequest()
              {
                  BucketName = bucketName,
                  Prefix = "outputfiles/"
              });

        if (outputFiles.S3Objects.Any())
        {
            var sampleOutput = outputFiles.S3Objects.OrderBy(s =>
s.LastModified).Last();
            Console.WriteLine($"\tOutput file: {sampleOutput.Key}");
            var outputSampleResponse = await _s3Client.GetObjectAsync(
                new GetObjectRequest()
                {
                    BucketName = bucketName,
                    Key = sampleOutput.Key
                });
```

```
            outputKey = sampleOutput.Key;
            StreamReader reader = new
 StreamReader(outputSampleResponse.ResponseStream);
            await reader.ReadLineAsync();
            Console.WriteLine("\tOutput file contents: \n");
            for (int i = 0; i < 10; i++)
            {
                if (!reader.EndOfStream)
                {
                    Console.WriteLine("\t" + await reader.ReadLineAsync());
                }
            }
        }

        Console.WriteLine(new string('-', 80));
        return outputKey;
    }

    /// <summary>
    /// Create a pipeline from the example pipeline JSON
    /// that includes the Lambda, callback, processing, and export jobs.
    /// </summary>
    /// <param name="roleArn">The ARN of the role for the pipeline.</param>
    /// <param name="functionArn">The ARN of the Lambda function for the pipeline.</
param>
    /// <param name="pipelineName">The name for the pipeline.</param>
    /// <returns>The ARN of the pipeline.</returns>
    public static async Task<string> SetupPipeline(string roleArn, string
 functionArn, string pipelineName)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Setting up the pipeline.");

        var pipelineJson = await File.ReadAllTextAsync("GeoSpatialPipeline.json");

        // Add the correct function ARN instead of the placeholder.
        pipelineJson = pipelineJson.Replace("*FUNCTION_ARN*", functionArn);

        var pipelineArn = await _sageMakerWrapper.SetupPipeline(pipelineJson,
 roleArn, pipelineName,
            "sdk example pipeline", pipelineName);

        Console.WriteLine($"\tPipeline set up with ARN {pipelineArn}.");
        Console.WriteLine(new string('-', 80));
```

```
        return pipelineArn;
    }

    /// <summary>
    /// Start a pipeline run with job configurations.
    /// </summary>
    /// <param name="queueUrl">The URL for the queue used in the pipeline.</param>
    /// <param name="roleArn">The ARN of the role.</param>
    /// <param name="pipelineName">The name of the pipeline.</param>
    /// <param name="bucketName">The name of the bucket.</param>
    /// <returns>The pipeline run ARN.</returns>
    public static async Task<string> ExecutePipeline(
        string queueUrl,
        string roleArn,
        string pipelineName,
        string bucketName)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Starting pipeline execution.");

        var input = $"s3://{bucketName}/samplefiles/latlongtest.csv";
        var output = $"s3://{bucketName}/outputfiles/";

        var executionARN =
            await _sageMakerWrapper.ExecutePipeline(queueUrl, input, output,
                pipelineName, roleArn);

        Console.WriteLine($"\tRun started with ARN {executionARN}.");
        Console.WriteLine(new string('-', 80));

        return executionARN;
    }

    /// <summary>
    /// Wait for a pipeline run to complete.
    /// </summary>
    /// <param name="executionArn">The pipeline run ARN.</param>
    /// <returns>Async task.</returns>
    public static async Task WaitForPipelineExecution(string executionArn)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Waiting for pipeline to finish.");
```

```
        PipelineExecutionStatus status;
        do
        {
            status = await
 _sageMakerWrapper.CheckPipelineExecutionStatus(executionArn);
            Thread.Sleep(30000);
            Console.WriteLine($"\tStatus is {status}.");
        } while (status == PipelineExecutionStatus.Executing);

        Console.WriteLine($"\tPipeline finished with status {status}.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <param name="askUser">True to ask the user for cleanup.</param>
    /// <param name="queueUrl">The URL of the queue to clean up.</param>
    /// <param name="pipelineName">The name of the pipeline.</param>
    /// <param name="bucketName">The name of the bucket.</param>
    /// <returns>Async task.</returns>
    public static async Task<bool> CleanupResources(
        bool askUser,
        string queueUrl,
        string pipelineName,
        string bucketName)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Clean up resources.");

        if (!askUser || GetYesNoResponse($"\tDelete pipeline {pipelineName}? (y/
n)"))
        {
            Console.WriteLine($"\tDeleting pipeline.");
            // Delete the pipeline.
            await _sageMakerWrapper.DeletePipelineByName(pipelineName);
        }

        if (!string.IsNullOrEmpty(queueUrl) && (!askUser ||
 GetYesNoResponse($"\tDelete queue {queueUrl}? (y/n)")))
        {
            Console.WriteLine($"\tDeleting queue.");
            // Delete the queue.
            await _sqsClient.DeleteQueueAsync(new DeleteQueueRequest(queueUrl));
```

```
        }

        if (!askUser || GetYesNoResponse($"\tDelete Amazon S3 bucket {bucketName}?
 (y/n)"))
        {
            Console.WriteLine($"\tDeleting bucket.");
            // Delete all objects in the bucket.
            var deleteList = await _s3Client.ListObjectsV2Async(new
 ListObjectsV2Request()
            {
                BucketName = bucketName
            });
            if (deleteList.KeyCount > 0)
            {
                await _s3Client.DeleteObjectsAsync(new DeleteObjectsRequest()
                {
                    BucketName = bucketName,
                    Objects = deleteList.S3Objects
                        .Select(o => new KeyVersion { Key = o.Key }).ToList()
                });
            }

            // Now delete the bucket.
            await _s3Client.DeleteBucketAsync(new DeleteBucketRequest()
            {
                BucketName = bucketName
            });
        }

        if (!askUser || GetYesNoResponse($"\tDelete lambda {lambdaFunctionName}? (y/
n)"))
        {
            Console.WriteLine($"\tDeleting lambda function.");

            await _lambdaClient.DeleteFunctionAsync(new DeleteFunctionRequest()
            {
                FunctionName = lambdaFunctionName
            });
        }

        if (!askUser || GetYesNoResponse($"\tDelete role {lambdaRoleName}? (y/n)"))
        {
            Console.WriteLine($"\tDetaching policies and deleting role.");
```

```
            foreach (var policy in lambdaRolePolicies)
            {
                await _iamClient!.DetachRolePolicyAsync(new
 DetachRolePolicyRequest()
                {
                    RoleName = lambdaRoleName,
                    PolicyArn = policy
                });
            }

            await _iamClient!.DeleteRoleAsync(new DeleteRoleRequest()
            {
                RoleName = lambdaRoleName
            });
        }

        if (!askUser || GetYesNoResponse($"\tDelete role {sageMakerRoleName}? (y/
n)"))
        {
            Console.WriteLine($"\tDetaching policies and deleting role.");

            foreach (var policy in sageMakerRolePolicies)
            {
                await _iamClient!.DetachRolePolicyAsync(new
 DetachRolePolicyRequest()
                {
                    RoleName = sageMakerRoleName,
                    PolicyArn = policy
                });
            }

            await _iamClient!.DeleteRoleAsync(new DeleteRoleRequest()
            {
                RoleName = sageMakerRoleName
            });
        }

        Console.WriteLine(new string('-', 80));
        return true;
    }

    /// <summary>
    /// Helper method to get a role's ARN if it already exists.
    /// </summary>
```

```
    /// <param name="roleName">The name of the AWS Identity and Access Management
 (IAM) Role to look for.</param>
    /// <returns>The role ARN if it exists, otherwise an empty string.</returns>
    private static async Task<string> GetRoleArnIfExists(string roleName)
    {
        Console.WriteLine($"Checking for role named {roleName}.");

        try
        {
            var existingRole = await _iamClient.GetRoleAsync(new GetRoleRequest()
            {
                RoleName = lambdaRoleName
            });
            return existingRole.Role.Arn;
        }
        catch (NoSuchEntityException)
        {
            return string.Empty;
        }
    }

    /// <summary>
    /// Helper method to 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;
    }
}
```

- 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。
  - CreatePipeline
  - DeletePipeline

- DescribePipelineExecution
- StartPipelineExecution
- UpdatePipeline

を使用した Secrets Manager の例 AWS SDK for .NET

次のコード例は、Secrets Manager AWS SDK for .NET で 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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。ここでは GitHub、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
- アクション

アクション

`GetSecretValue`

次の例は、`GetSecretValue` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

```
using Amazon.SecretsManager;
using Amazon.SecretsManager.Model;

/// <summary>
/// This example uses the Amazon Web Service Secrets Manager to retrieve
/// the secret value for the provided secret name.
/// </summary>
public class GetSecretValue
{
    /// <summary>
    /// The main method initializes the necessary values and then calls
    /// the GetSecretAsync and DecodeString methods to get the decoded
    /// secret value for the secret named in secretName.
    /// </summary>
    public static async Task Main()
    {
        string secretName = "<<{{MySecretName}}>>";
        string secret;

        IAmazonSecretsManager client = new AmazonSecretsManagerClient();

        var response = await GetSecretAsync(client, secretName);

        if (response is not null)
        {
            secret = DecodeString(response);

            if (!string.IsNullOrEmpty(secret))
            {
                Console.WriteLine($"The decoded secret value is: {secret}.");
            }
            else
            {
                Console.WriteLine("No secret value was returned.");
            }
        }
    }

    /// <summary>
    /// Retrieves the secret value given the name of the secret to
    /// retrieve.
    /// </summary>
    /// <param name="client">The client object used to retrieve the secret
    /// value for the given secret name.</param>
```

```
        /// <param name="secretName">The name of the secret value to retrieve.</
param>
        /// <returns>The GetSecretValueReponse object returned by
        /// GetSecretValueAsync.</returns>
        public static async Task<GetSecretValueResponse> GetSecretAsync(
            IAmazonSecretsManager client,
            string secretName)
        {
            GetSecretValueRequest request = new GetSecretValueRequest()
            {
                SecretId = secretName,
                VersionStage = "AWSCURRENT", // VersionStage defaults to AWSCURRENT
 if unspecified.
            };

            GetSecretValueResponse response = null;

            // For the sake of simplicity, this example handles only the most
            // general SecretsManager exception.
            try
            {
                response = await client.GetSecretValueAsync(request);
            }
            catch (AmazonSecretsManagerException e)
            {
                Console.WriteLine($"Error: {e.Message}");
            }

            return response;
        }

        /// <summary>
        /// Decodes the secret returned by the call to GetSecretValueAsync and
        /// returns it to the calling program.
        /// </summary>
        /// <param name="response">A GetSecretValueResponse object containing
        /// the requested secret value returned by GetSecretValueAsync.</param>
        /// <returns>A string representing the decoded secret value.</returns>
        public static string DecodeString(GetSecretValueResponse response)
        {
            // Decrypts secret using the associated AWS Key Management Service
            // Customer Master Key (CMK.) Depending on whether the secret is a
            // string or binary value, one of these fields will be populated.
            if (response.SecretString is not null)
```

```
            {
                var secret = response.SecretString;
                return secret;
            }
            else if (response.SecretBinary is not null)
            {
                var memoryStream = response.SecretBinary;
                StreamReader reader = new StreamReader(memoryStream);
                string decodedBinarySecret =
System.Text.Encoding.UTF8.GetString(Convert.FromBase64String(reader.ReadToEnd()));
                return decodedBinarySecret;
            }
            else
            {
                return string.Empty;
            }
        }
    }
```

- API の詳細については、「 API リファレンスGetSecretValue」の「」を参照してください。 AWS SDK for .NET

を使用した Amazon SES の例 AWS SDK for .NET

次のコード例は、Amazon SES AWS SDK for .NET で 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。 Amazon SES

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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。ここでは GitHub、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

- アクション

アクション

`CreateTemplate`

次の例は、CreateTemplate を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
   /// <summary>
   /// Create an email template.
   /// </summary>
   /// <param name="name">Name of the template.</param>
   /// <param name="subject">Email subject.</param>
   /// <param name="text">Email body text.</param>
   /// <param name="html">Email HTML body text.</param>
   /// <returns>True if successful.</returns>
   public async Task<bool> CreateEmailTemplateAsync(string name, string subject,
string text,
       string html)
   {
       var success = false;
       try
       {
           var response = await _amazonSimpleEmailService.CreateTemplateAsync(
               new CreateTemplateRequest
               {
                   Template = new Template
                   {
                       TemplateName = name,
                       SubjectPart = subject,
                       TextPart = text,
                       HtmlPart = html
                   }
               });
           success = response.HttpStatusCode == HttpStatusCode.OK;
       }
```

```
        catch (Exception ex)
        {
            Console.WriteLine("CreateEmailTemplateAsync failed with exception: " +
ex.Message);
        }

        return success;
    }
```

- API の詳細については、「 API リファレンスCreateTemplate」の「」を参照してください。AWS SDK for .NET

`DeleteIdentity`

次の例は、`DeleteIdentity` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete an email identity.
    /// </summary>
    /// <param name="identityEmail">The identity email to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteIdentityAsync(string identityEmail)
    {
        var success = false;
        try
        {
            var response = await _amazonSimpleEmailService.DeleteIdentityAsync(
                new DeleteIdentityRequest
                {
                    Identity = identityEmail
```

```
                });
            success = response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (Exception ex)
        {
            Console.WriteLine("DeleteIdentityAsync failed with exception: " +
ex.Message);
        }

        return success;
    }
```

- API の詳細については、「 API リファレンスDeleteIdentity」の「」を参照してください。 AWS SDK for .NET

`DeleteTemplate`

次の例は、DeleteTemplate を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete an email template.
    /// </summary>
    /// <param name="templateName">Name of the template.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteEmailTemplateAsync(string templateName)
    {
        var success = false;
        try
        {
            var response = await _amazonSimpleEmailService.DeleteTemplateAsync(
```

```
                new DeleteTemplateRequest
                {
                    TemplateName = templateName
                });
            success = response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (Exception ex)
        {
            Console.WriteLine("DeleteEmailTemplateAsync failed with exception: " +
ex.Message);
        }

        return success;
    }
```

- API の詳細については、「 API リファレンスDeleteTemplate」の「」を参照してください。 AWS SDK for .NET

`GetIdentityVerificationAttributes`

次の例は、`GetIdentityVerificationAttributes` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get identity verification status for an email.
    /// </summary>
    /// <returns>The verification status of the email.</returns>
    public async Task<VerificationStatus> GetIdentityStatusAsync(string email)
    {
        var result = VerificationStatus.TemporaryFailure;
        try
```

```
        {
            var response =
                await
_amazonSimpleEmailService.GetIdentityVerificationAttributesAsync(
                    new GetIdentityVerificationAttributesRequest
                    {
                        Identities = new List<string> { email }
                    });

            if (response.VerificationAttributes.ContainsKey(email))
                result = response.VerificationAttributes[email].VerificationStatus;
        }
        catch (Exception ex)
        {
            Console.WriteLine("GetIdentityStatusAsync failed with exception: " +
ex.Message);
        }

        return result;
    }
```

- API の詳細については、「 API リファレンスGetIdentityVerificationAttributes」の「」を参照してください。 AWS SDK for .NET

`GetSendQuota`

次の例は、`GetSendQuota` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get information on the current account's send quota.
```

```
    /// </summary>
    /// <returns>The send quota response data.</returns>
    public async Task<GetSendQuotaResponse> GetSendQuotaAsync()
    {
        var result = new GetSendQuotaResponse();
        try
        {
            var response = await _amazonSimpleEmailService.GetSendQuotaAsync(
                new GetSendQuotaRequest());
            result = response;
        }
        catch (Exception ex)
        {
            Console.WriteLine("GetSendQuotaAsync failed with exception: " +
ex.Message);
        }

        return result;
    }
```

- API の詳細については、「 API リファレンスGetSendQuota」の「」を参照してください。AWS SDK for .NET

`ListIdentities`

次の例は、`ListIdentities` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get the identities of a specified type for the current account.
    /// </summary>
```

```
    /// <param name="identityType">IdentityType to list.</param>
    /// <returns>The list of identities.</returns>
    public async Task<List<string>> ListIdentitiesAsync(IdentityType identityType)
    {
        var result = new List<string>();
        try
        {
            var response = await _amazonSimpleEmailService.ListIdentitiesAsync(
                new ListIdentitiesRequest
                {
                    IdentityType = identityType
                });
            result = response.Identities;
        }
        catch (Exception ex)
        {
            Console.WriteLine("ListIdentitiesAsync failed with exception: " +
ex.Message);
        }

        return result;
    }
```

- API の詳細については、「 API リファレンスListIdentities」の「」を参照してください。 AWS SDK for .NET

`ListTemplates`

次の例は、`ListTemplates` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// List email templates for the current account.
    /// </summary>
    /// <returns>A list of template metadata.</returns>
    public async Task<List<TemplateMetadata>> ListEmailTemplatesAsync()
    {
        var result = new List<TemplateMetadata>();
        try
        {
            var response = await _amazonSimpleEmailService.ListTemplatesAsync(
                new ListTemplatesRequest());
            result = response.TemplatesMetadata;
        }
        catch (Exception ex)
        {
            Console.WriteLine("ListEmailTemplatesAsync failed with exception: " +
ex.Message);
        }

        return result;
    }
```

- API の詳細については、「 API リファレンスListTemplates」の「」を参照してください。 AWS SDK for .NET

`SendEmail`

次の例は、SendEmail を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
```

```
    ///  Send an email by using Amazon SES.
    /// </summary>
    /// <param name="toAddresses">List of recipients.</param>
    /// <param name="ccAddresses">List of cc recipients.</param>
    /// <param name="bccAddresses">List of bcc recipients.</param>
    /// <param name="bodyHtml">Body of the email in HTML.</param>
    /// <param name="bodyText">Body of the email in plain text.</param>
    /// <param name="subject">Subject line of the email.</param>
    /// <param name="senderAddress">From address.</param>
    /// <returns>The messageId of the email.</returns>
    public async Task<string> SendEmailAsync(List<string> toAddresses,
        List<string> ccAddresses, List<string> bccAddresses,
        string bodyHtml, string bodyText, string subject, string senderAddress)
    {
        var messageId = "";
        try
        {
            var response = await _amazonSimpleEmailService.SendEmailAsync(
                new SendEmailRequest
                {
                    Destination = new Destination
                    {
                        BccAddresses = bccAddresses,
                        CcAddresses = ccAddresses,
                        ToAddresses = toAddresses
                    },
                    Message = new Message
                    {
                        Body = new Body
                        {
                            Html = new Content
                            {
                                Charset = "UTF-8",
                                Data = bodyHtml
                            },
                            Text = new Content
                            {
                                Charset = "UTF-8",
                                Data = bodyText
                            }
                        },
                        Subject = new Content
                        {
                            Charset = "UTF-8",
```

```
                        Data = subject
                    }
                },
                Source = senderAddress
            });
        messageId = response.MessageId;
    }
    catch (Exception ex)
    {
        Console.WriteLine("SendEmailAsync failed with exception: " +
ex.Message);
    }

    return messageId;
}
```

- API の詳細については、「 API リファレンスSendEmail」の「」を参照してください。 AWS SDK for .NET

SendTemplatedEmail

次の例は、SendTemplatedEmail を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
/// Send an email using a template.
/// </summary>
/// <param name="sender">Address of the sender.</param>
/// <param name="recipients">Addresses of the recipients.</param>
/// <param name="templateName">Name of the email template.</param>
/// <param name="templateDataObject">Data for the email template.</param>
```

```
   /// <returns>The messageId of the email.</returns>
   public async Task<string> SendTemplateEmailAsync(string sender, List<string>
recipients,
       string templateName, object templateDataObject)
   {
       var messageId = "";
       try
       {
           // Template data should be serialized JSON from either a class or a
dynamic object.
           var templateData = JsonSerializer.Serialize(templateDataObject);

           var response = await _amazonSimpleEmailService.SendTemplatedEmailAsync(
               new SendTemplatedEmailRequest
               {
                   Source = sender,
                   Destination = new Destination
                   {
                       ToAddresses = recipients
                   },
                   Template = templateName,
                   TemplateData = templateData
               });
           messageId = response.MessageId;
       }
       catch (Exception ex)
       {
           Console.WriteLine("SendTemplateEmailAsync failed with exception: " +
ex.Message);
       }

       return messageId;
   }
```

- API の詳細については、「 API リファレンスSendTemplatedEmail」の「」を参照してください。 AWS SDK for .NET

VerifyEmailIdentity

次の例は、VerifyEmailIdentity を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Starts verification of an email identity. This request sends an email
    /// from Amazon SES to the specified email address. To complete
    /// verification, follow the instructions in the email.
    /// </summary>
    /// <param name="recipientEmailAddress">Email address to verify.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> VerifyEmailIdentityAsync(string recipientEmailAddress)
    {
        var success = false;
        try
        {
            var response = await _amazonSimpleEmailService.VerifyEmailIdentityAsync(
                new VerifyEmailIdentityRequest
                {
                    EmailAddress = recipientEmailAddress
                });

            success = response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (Exception ex)
        {
            Console.WriteLine("VerifyEmailIdentityAsync failed with exception: " +
ex.Message);
        }

        return success;
    }
```

- API の詳細については、「 API リファレンスVerifyEmailIdentity」の「」を参照してください。AWS SDK for .NET

を使用した Amazon SES API v2 の例 AWS SDK for .NET

次のコード例は、Amazon SES API v2 AWS SDK for .NET で 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。 Amazon SES

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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。ここでは GitHub、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

アクション

`CreateContact`

次の例は、`CreateContact` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
/// Creates a contact and adds it to the specified contact list.
/// </summary>
/// <param name="emailAddress">The email address of the contact.</param>
/// <param name="contactListName">The name of the contact list.</param>
/// <returns>The response from the CreateContact operation.</returns>
```

```
   public async Task<bool> CreateContactAsync(string emailAddress, string
contactListName)
   {
       var request = new CreateContactRequest
       {
           EmailAddress = emailAddress,
           ContactListName = contactListName
       };

       try
       {
           var response = await _sesClient.CreateContactAsync(request);
           return response.HttpStatusCode == HttpStatusCode.OK;
       }
       catch (AlreadyExistsException ex)
       {
           Console.WriteLine($"Contact with email address {emailAddress} already
exists in the contact list {contactListName}.");
           Console.WriteLine(ex.Message);
           return true;
       }
       catch (NotFoundException ex)
       {
           Console.WriteLine($"The contact list {contactListName} does not
exist.");
           Console.WriteLine(ex.Message);
       }
       catch (TooManyRequestsException ex)
       {
           Console.WriteLine("Too many requests were made. Please try again
later.");
           Console.WriteLine(ex.Message);
       }
       catch (Exception ex)
       {
           Console.WriteLine($"An error occurred while creating the contact:
{ex.Message}");
       }
       return false;
   }
```

- API の詳細については、「 API リファレンスCreateContact」の「」を参照してください。AWS SDK for .NET

`CreateContactList`

次の例は、CreateContactList を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Creates a contact list with the specified name.
    /// </summary>
    /// <param name="contactListName">The name of the contact list.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> CreateContactListAsync(string contactListName)
    {
        var request = new CreateContactListRequest
        {
            ContactListName = contactListName
        };

        try
        {
            var response = await _sesClient.CreateContactListAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (AlreadyExistsException ex)
        {
            Console.WriteLine($"Contact list with name {contactListName} already
exists.");
            Console.WriteLine(ex.Message);
            return true;
        }
        catch (LimitExceededException ex)
        {
            Console.WriteLine("The limit for contact lists has been exceeded.");
```

```
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating the contact list:
{ex.Message}");
        }
        return false;
    }
```

- API の詳細については、「 API リファレンスCreateContactList」の「」を参照してください。AWS SDK for .NET

`CreateEmailIdentity`

次の例は、`CreateEmailIdentity` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Creates an email identity (email address or domain) and starts the
verification process.
    /// </summary>
    /// <param name="emailIdentity">The email address or domain to create and
verify.</param>
    /// <returns>The response from the CreateEmailIdentity operation.</returns>
    public async Task<CreateEmailIdentityResponse> CreateEmailIdentityAsync(string
emailIdentity)
```

```
    {
        var request = new CreateEmailIdentityRequest
        {
            EmailIdentity = emailIdentity
        };

        try
        {
            var response = await _sesClient.CreateEmailIdentityAsync(request);
            return response;
        }
        catch (AlreadyExistsException ex)
        {
            Console.WriteLine($"Email identity {emailIdentity} already exists.");
            Console.WriteLine(ex.Message);
            throw;
        }
        catch (ConcurrentModificationException ex)
        {
            Console.WriteLine($"The email identity {emailIdentity} is being modified
by another operation or thread.");
            Console.WriteLine(ex.Message);
            throw;
        }
        catch (LimitExceededException ex)
        {
            Console.WriteLine("The limit for email identities has been exceeded.");
            Console.WriteLine(ex.Message);
            throw;
        }
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The email identity {emailIdentity} does not
exist.");
            Console.WriteLine(ex.Message);
            throw;
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
            throw;
        }
```

```
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating the email identity:
{ex.Message}");
            throw;
        }
    }
```

- API の詳細については、「 API リファレンスCreateEmailIdentity」の「」を参照してください。 AWS SDK for .NET

`CreateEmailTemplate`

次の例は、`CreateEmailTemplate` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Creates an email template with the specified content.
    /// </summary>
    /// <param name="templateName">The name of the email template.</param>
    /// <param name="subject">The subject of the email template.</param>
    /// <param name="htmlContent">The HTML content of the email template.</param>
    /// <param name="textContent">The text content of the email template.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> CreateEmailTemplateAsync(string templateName, string
subject, string htmlContent, string textContent)
    {
        var request = new CreateEmailTemplateRequest
        {
            TemplateName = templateName,
            TemplateContent = new EmailTemplateContent
            {
                Subject = subject,
```

```
                Html = htmlContent,
                Text = textContent
            }
        };

        try
        {
            var response = await _sesClient.CreateEmailTemplateAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (AlreadyExistsException ex)
        {
            Console.WriteLine($"Email template with name {templateName} already
exists.");
            Console.WriteLine(ex.Message);
        }
        catch (LimitExceededException ex)
        {
            Console.WriteLine("The limit for email templates has been exceeded.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating the email template:
{ex.Message}");
        }

        return false;
    }
```

- API の詳細については、「 API リファレンスCreateEmailTemplate」の「」を参照してください。 AWS SDK for .NET

DeleteContactList

次の例は、DeleteContactList を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Deletes a contact list and all contacts within it.
    /// </summary>
    /// <param name="contactListName">The name of the contact list to delete.</
param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteContactListAsync(string contactListName)
    {
        var request = new DeleteContactListRequest
        {
            ContactListName = contactListName
        };

        try
        {
            var response = await _sesClient.DeleteContactListAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (ConcurrentModificationException ex)
        {
            Console.WriteLine($"The contact list {contactListName} is being modified
 by another operation or thread.");
            Console.WriteLine(ex.Message);
        }
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The contact list {contactListName} does not
 exist.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
 later.");
```

```
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting the contact list:
{ex.Message}");
        }

        return false;
    }
```

- API の詳細については、「 API リファレンスDeleteContactList」の「」を参照してください。 AWS SDK for .NET

`DeleteEmailIdentity`

次の例は、DeleteEmailIdentity を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Deletes an email identity (email address or domain).
    /// </summary>
    /// <param name="emailIdentity">The email address or domain to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteEmailIdentityAsync(string emailIdentity)
    {
        var request = new DeleteEmailIdentityRequest
        {
            EmailIdentity = emailIdentity
        };

        try
        {
```

```
            var response = await _sesClient.DeleteEmailIdentityAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (ConcurrentModificationException ex)
        {
            Console.WriteLine($"The email identity {emailIdentity} is being modified
by another operation or thread.");
            Console.WriteLine(ex.Message);
        }
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The email identity {emailIdentity} does not
exist.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting the email identity:
{ex.Message}");
        }

        return false;
    }
```

- API の詳細については、「 API リファレンスDeleteEmailIdentity」の「」を参照してください。 AWS SDK for .NET

DeleteEmailTemplate

次の例は、DeleteEmailTemplate を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Deletes an email template.
    /// </summary>
    /// <param name="templateName">The name of the email template to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteEmailTemplateAsync(string templateName)
    {
        var request = new DeleteEmailTemplateRequest
        {
            TemplateName = templateName
        };

        try
        {
            var response = await _sesClient.DeleteEmailTemplateAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The email template {templateName} does not exist.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting the email template:
{ex.Message}");
        }
```

```
        return false;
    }
```

- API の詳細については、「 API リファレンスDeleteEmailTemplate」の「」を参照してください。 AWS SDK for .NET

`ListContacts`

次の例は、`ListContacts` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Lists the contacts in the specified contact list.
    /// </summary>
    /// <param name="contactListName">The name of the contact list.</param>
    /// <returns>The list of contacts response from the ListContacts operation.</
returns>
    public async Task<List<Contact>> ListContactsAsync(string contactListName)
    {
        var request = new ListContactsRequest
        {
            ContactListName = contactListName
        };

        try
        {
            var response = await _sesClient.ListContactsAsync(request);
            return response.Contacts;
        }
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The contact list {contactListName} does not
 exist.");
```

```
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while listing the contacts:
{ex.Message}");
        }

        return new List<Contact>();
    }
```

- API の詳細については、「 API リファレンスListContacts」の「」を参照してください。 AWS SDK for .NET

`SendEmail`

次の例は、SendEmail を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Sends an email with the specified content and options.
    /// </summary>
    /// <param name="fromEmailAddress">The email address to send the email from.</
param>
    /// <param name="toEmailAddresses">The email addresses to send the email to.</
param>
    /// <param name="subject">The subject of the email.</param>
```

```
   /// <param name="htmlContent">The HTML content of the email.</param>
   /// <param name="textContent">The text content of the email.</param>
   /// <param name="templateName">The name of the email template to use
(optional).</param>
   /// <param name="templateData">The data to replace placeholders in the email
template (optional).</param>
   /// <param name="contactListName">The name of the contact list for unsubscribe
functionality (optional).</param>
   /// <returns>The MessageId response from the SendEmail operation.</returns>
   public async Task<string> SendEmailAsync(string fromEmailAddress, List<string>
toEmailAddresses, string? subject,
       string? htmlContent, string? textContent, string? templateName = null,
string? templateData = null, string? contactListName = null)
   {
       var request = new SendEmailRequest
       {
           FromEmailAddress = fromEmailAddress
       };

       if (toEmailAddresses.Any())
       {
           request.Destination = new Destination { ToAddresses =
toEmailAddresses };
       }

       if (!string.IsNullOrEmpty(templateName))
       {
           request.Content = new EmailContent()
           {
               Template = new Template
               {
                   TemplateName = templateName,
                   TemplateData = templateData
               }
           };
       }
       else
       {
           request.Content = new EmailContent
           {
               Simple = new Message
               {
                   Subject = new Content { Data = subject },
                   Body = new Body
```

```
                    {
                        Html = new Content { Data = htmlContent },
                        Text = new Content { Data = textContent }
                    }
                }
            };
        }

        if (!string.IsNullOrEmpty(contactListName))
        {
            request.ListManagementOptions = new ListManagementOptions
            {
                ContactListName = contactListName
            };
        }

        try
        {
            var response = await _sesClient.SendEmailAsync(request);
            return response.MessageId;
        }
        catch (AccountSuspendedException ex)
        {
            Console.WriteLine("The account's ability to send email has been
permanently restricted.");
            Console.WriteLine(ex.Message);
        }
        catch (MailFromDomainNotVerifiedException ex)
        {
            Console.WriteLine("The sending domain is not verified.");
            Console.WriteLine(ex.Message);
        }
        catch (MessageRejectedException ex)
        {
            Console.WriteLine("The message content is invalid.");
            Console.WriteLine(ex.Message);
        }
        catch (SendingPausedException ex)
        {
            Console.WriteLine("The account's ability to send email is currently
paused.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
```

```
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while sending the email:
{ex.Message}");
        }

        return string.Empty;
    }
```

- API の詳細については、「 API リファレンスSendEmail」の「」を参照してください。 AWS SDK for .NET

シナリオ

ニュースレターワークフロー

次のコード例は、Amazon SES API v2 ニュースレターワークフローの方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ワークフローを実行します。

```
using System.Diagnostics;
using System.Text.RegularExpressions;
using Amazon.SimpleEmailV2;
using Amazon.SimpleEmailV2.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
```

```
using Microsoft.Extensions.Logging.Debug;

namespace Sesv2Scenario;

public static class NewsletterWorkflow
{
    /*
      This workflow demonstrates how to use the Amazon Simple Email Service (SES) v2
 to send a coupon newsletter to a list of subscribers.
      The workflow performs the following tasks:

      1. Prepare the application:
         - Create a verified email identity for sending and replying to emails.
         - Create a contact list to store the subscribers' email addresses.
         - Create an email template for the coupon newsletter.

      2. Gather subscriber email addresses:
         - Prompt the user for a base email address.
         - Create 3 variants of the email address using subaddress extensions (e.g.,
 user+ses-weekly-newsletter-1@example.com).
         - Add each variant as a contact to the contact list.
         - Send a welcome email to each new contact.

      3. Send the coupon newsletter:
         - Retrieve the list of contacts from the contact list.
         - Send the coupon newsletter using the email template to each contact.

      4. Monitor and review:
         - Provide instructions for the user to review the sending activity and
 metrics in the AWS console.

      5. Clean up resources:
         - Delete the contact list (which also deletes all contacts within it).
         - Delete the email template.
         - Optionally delete the verified email identity.

    */

    public static SESv2Wrapper _sesv2Wrapper;
    public static string? _baseEmailAddress = null;
    public static string? _verifiedEmail = null;
    private static string _contactListName = "weekly-coupons-newsletter";
    private static string _templateName = "weekly-coupons";
    private static string _subject = "Weekly Coupons Newsletter";
```

```
    private static string _htmlContentFile = "coupon-newsletter.html";
    private static string _textContentFile = "coupon-newsletter.txt";
    private static string _htmlWelcomeFile = "welcome.html";
    private static string _textWelcomeFile = "welcome.txt";
    private static string _couponsDataFile = "sample_coupons.json";

    // Relative location of the shared workflow resources folder.
    private static string _resourcesFilePathLocation = "../../../../../../../
workflows/sesv2_weekly_mailer/resources/";

    public 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<IAmazonSimpleEmailServiceV2>()
                    .AddTransient<SESv2Wrapper>()
            )
            .Build();

        ServicesSetup(host);

        try
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Welcome to the Amazon SES v2 Coupon Newsletter
 Workflow.");
            Console.WriteLine("This workflow demonstrates how to use the Amazon
 Simple Email Service (SES) v2 " +
                              "\r\nto send a coupon newsletter to a list of
 subscribers.");

            // Prepare the application.
            var emailIdentity = await PrepareApplication();

            // Gather subscriber email addresses.
            await GatherSubscriberEmailAddresses(emailIdentity);
```

```
            // Send the coupon newsletter.
            await SendCouponNewsletter(emailIdentity);

            // Monitor and review.
            MonitorAndReview(true);

            // Clean up resources.
            await Cleanup(emailIdentity, true);

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Amazon SES v2 Coupon Newsletter Workflow is
complete.");
            Console.WriteLine(new string('-', 80));
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred: {ex.Message}");
        }
    }

    /// <summary>
    /// Populate the services for use within the console application.
    /// </summary>
    /// <param name="host">The services host.</param>
    private static void ServicesSetup(IHost host)
    {
        _sesv2Wrapper = host.Services.GetRequiredService<SESv2Wrapper>();
    }

    /// <summary>
    /// Set up the resources for the workflow.
    /// </summary>
    /// <returns>The email address of the verified identity.</returns>
    public static async Task<string?> PrepareApplication()
    {
        var htmlContent = await File.ReadAllTextAsync(_resourcesFilePathLocation +
_htmlContentFile);
        var textContent = await File.ReadAllTextAsync(_resourcesFilePathLocation +
_textContentFile);

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("1. In this step, we will prepare the application:" +
```

```
                          "\r\n  - Create a verified email identity for sending and
replying to emails." +
                          "\r\n  - Create a contact list to store the subscribers'
email addresses." +
                          "\r\n  - Create an email template for the coupon
newsletter.\r\n");

        // Prompt the user for a verified email address.
        while (!IsEmail(_verifiedEmail))
        {
            Console.Write("Enter a verified email address or an email to verify: ");
            _verifiedEmail = Console.ReadLine();
        }

        try
        {
            // Create an email identity and start the verification process.
            await _sesv2Wrapper.CreateEmailIdentityAsync(_verifiedEmail);
            Console.WriteLine($"Identity {_verifiedEmail} created.");
        }
        catch (AlreadyExistsException)
        {
            Console.WriteLine($"Identity {_verifiedEmail} already exists.");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error creating email identity: {ex.Message}");
        }

        // Create a contact list.
        try
        {
            await _sesv2Wrapper.CreateContactListAsync(_contactListName);
            Console.WriteLine($"Contact list {_contactListName} created.");
        }
        catch (AlreadyExistsException)
        {
            Console.WriteLine($"Contact list {_contactListName} already exists.");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error creating contact list: {ex.Message}");
        }
```

```
        // Create an email template.
        try
        {
            await _sesv2Wrapper.CreateEmailTemplateAsync(_templateName, _subject,
htmlContent, textContent);
            Console.WriteLine($"Email template {_templateName} created.");
        }
        catch (AlreadyExistsException)
        {
            Console.WriteLine($"Email template {_templateName} already exists.");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error creating email template: {ex.Message}");
        }

        return _verifiedEmail;
    }

    /// <summary>
    /// Generate subscriber addresses and send welcome emails.
    /// </summary>
    /// <param name="fromEmailAddress">The verified email address from
PrepareApplication.</param>
    /// <returns>True if successful.</returns>
    public static async Task<bool> GatherSubscriberEmailAddresses(string
fromEmailAddress)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("2. In Step 2, we will gather subscriber email addresses:"
+
                          "\r\n  - Prompt the user for a base email address." +
                          "\r\n  - Create 3 variants of the email address using
subaddress extensions (e.g., user+ses-weekly-newsletter-1@example.com)." +
                          "\r\n  - Add each variant as a contact to the contact
list." +
                          "\r\n  - Send a welcome email to each new contact.\r\n");

        // Prompt the user for a base email address.
        while (!IsEmail(_baseEmailAddress))
        {
            Console.Write("Enter a base email address (e.g., user@example.com): ");
            _baseEmailAddress = Console.ReadLine();
        }
```

```
        // Create 3 variants of the email address using +ses-weekly-newsletter-1,
 +ses-weekly-newsletter-2, etc.
        var baseEmailAddressParts = _baseEmailAddress!.Split("@");
        for (int i = 1; i <= 3; i++)
        {
            string emailAddress = $"{baseEmailAddressParts[0]}+ses-weekly-
newsletter-{i}@{baseEmailAddressParts[1]}";

            try
            {
                // Create a contact with the email address in the contact list.
                await _sesv2Wrapper.CreateContactAsync(emailAddress,
 _contactListName);
                Console.WriteLine($"Contact {emailAddress} added to the
 {_contactListName} contact list.");
            }
            catch (AlreadyExistsException)
            {
                Console.WriteLine($"Contact {emailAddress} already exists in the
 {_contactListName} contact list.");
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Error creating contact {emailAddress}:
 {ex.Message}");
                return false;
            }

            // Send a welcome email to the new contact.
            try
            {
                string subject = "Welcome to the Weekly Coupons Newsletter";
                string htmlContent = await
 File.ReadAllTextAsync(_resourcesFilePathLocation + _htmlWelcomeFile);
                string textContent = await
 File.ReadAllTextAsync(_resourcesFilePathLocation + _textWelcomeFile);

                await _sesv2Wrapper.SendEmailAsync(fromEmailAddress, new
 List<string> { emailAddress }, subject, htmlContent, textContent);
                Console.WriteLine($"Welcome email sent to {emailAddress}.");
            }
            catch (Exception ex)
            {
```

```
                Console.WriteLine($"Error sending welcome email to {emailAddress}:
{ex.Message}");
                return false;
            }

            // Wait 2 seconds before sending the next email (if the account is in
the SES Sandbox).
            await Task.Delay(2000);
        }

        return true;
    }

    /// <summary>
    ///  Send the coupon newsletter to the subscribers in the contact list.
    /// </summary>
    /// <param name="fromEmailAddress">The verified email address from
PrepareApplication.</param>
    /// <returns>True if successful.</returns>
    public static async Task<bool> SendCouponNewsletter(string fromEmailAddress)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("3. In this step, we will send the coupon newsletter:" +
                          "\r\n  - Retrieve the list of contacts from the contact
list." +
                          "\r\n  - Send the coupon newsletter using the email
template to each contact.\r\n");


        // Retrieve the list of contacts from the contact list.
        var contacts = await _sesv2Wrapper.ListContactsAsync(_contactListName);
        if (!contacts.Any())
        {
            Console.WriteLine($"No contacts found in the {_contactListName} contact
list.");
            return false;
        }

        // Load the coupon data from the sample_coupons.json file.
        string couponsData = await File.ReadAllTextAsync(_resourcesFilePathLocation
+ _couponsDataFile);

        // Send the coupon newsletter to each contact using the email template.
        try
```

```
        {
            foreach (var contact in contacts)
            {
                // To use the Contact List for list management, send to only one
address at a time.
                await _sesv2Wrapper.SendEmailAsync(fromEmailAddress,
                    new List<string> { contact.EmailAddress },
                    null, null, null, _templateName, couponsData, _contactListName);
            }

            Console.WriteLine($"Coupon newsletter sent to contact list
{_contactListName}.");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error sending coupon newsletter to contact list
{_contactListName}: {ex.Message}");
            return false;
        }

        return true;
    }

    /// <summary>
    /// Provide instructions for monitoring sending activity and metrics.
    /// </summary>
    /// <param name="interactive">True to run in interactive mode.</param>
    /// <returns>True if successful.</returns>
    public static bool MonitorAndReview(bool interactive)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("4. In step 4, we will monitor and review:" +
                          "\r\n  - Provide instructions for the user to review the
sending activity and metrics in the AWS console.\r\n");

        Console.WriteLine("Review your sending activity using the SES Homepage in
the AWS console.");
        Console.WriteLine("Press Enter to open the SES Homepage in your default
browser...");
        if (interactive)
        {
            Console.ReadLine();
            try
            {
```

```
                // Open the SES Homepage in the default browser.
                Process.Start(new ProcessStartInfo
                {
                    FileName = "https://console.aws.amazon.com/ses/home",
                    UseShellExecute = true
                });
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Error opening the SES Homepage: {ex.Message}");
                return false;
            }
        }

        Console.WriteLine("Review the sending activity and email metrics, then press
 Enter to continue...");
        if (interactive)
            Console.ReadLine();
        return true;
    }

    /// <summary>
    /// Clean up the resources used in the workflow.
    /// </summary>
    /// <param name="verifiedEmailAddress">The verified email address from
 PrepareApplication.</param>
    /// <param name="interactive">True if interactive.</param>
    /// <returns>Async task.</returns>
    public static async Task<bool> Cleanup(string verifiedEmailAddress, bool
 interactive)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("5. Finally, we clean up resources:" +
                          "\r\n  - Delete the contact list (which also deletes all
 contacts within it)." +
                          "\r\n  - Delete the email template." +
                          "\r\n  - Optionally delete the verified email identity.\r
\n");

        Console.WriteLine("Cleaning up resources...");

        // Delete the contact list (this also deletes all contacts in the list).
        try
        {
```

```
            await _sesv2Wrapper.DeleteContactListAsync(_contactListName);
            Console.WriteLine($"Contact list {_contactListName} deleted.");
        }
        catch (NotFoundException)
        {
            Console.WriteLine($"Contact list {_contactListName} not found.");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error deleting contact list {_contactListName}:
{ex.Message}");
            return false;
        }

        // Delete the email template.
        try
        {
            await _sesv2Wrapper.DeleteEmailTemplateAsync(_templateName);
            Console.WriteLine($"Email template {_templateName} deleted.");
        }
        catch (NotFoundException)
        {
            Console.WriteLine($"Email template {_templateName} not found.");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error deleting email template {_templateName}:
{ex.Message}");
            return false;
        }

        // Ask the user if they want to delete the email identity.
        var deleteIdentity = !interactive ||
            GetYesNoResponse(
                $"Do you want to delete the email identity {verifiedEmailAddress}?
(y/n) ");
        if (deleteIdentity)
        {
            try
            {
                await _sesv2Wrapper.DeleteEmailIdentityAsync(verifiedEmailAddress);
                Console.WriteLine($"Email identity {verifiedEmailAddress}
deleted.");
            }
```

```
            catch (NotFoundException)
            {
                Console.WriteLine(
                    $"Email identity {verifiedEmailAddress} not found.");
            }
            catch (Exception ex)
            {
                Console.WriteLine(
                    $"Error deleting email identity {verifiedEmailAddress}:
{ex.Message}");
                return false;
            }
        }
        else
        {
            Console.WriteLine(
                $"Skipping deletion of email identity {verifiedEmailAddress}.");
        }

        return true;
    }

    /// <summary>
    /// Helper method to 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;
    }

    /// <summary>
    /// Simple check to verify a string is an email address.
    /// </summary>
    /// <param name="email">The string to verify.</param>
    /// <returns>True if a valid email.</returns>
    private static bool IsEmail(string? email)
    {
        if (string.IsNullOrEmpty(email))
```

```
            return false;
        return Regex.IsMatch(email, @"^[^@\s]+@[^@\s]+\.[^@\s]+$",
 RegexOptions.IgnoreCase);
    }
}
```

サービスオペレーションのラッパー。

```
using System.Net;
using Amazon.SimpleEmailV2;
using Amazon.SimpleEmailV2.Model;

namespace Sesv2Scenario;

/// <summary>
/// Wrapper class for Amazon Simple Email Service (SES) v2 operations.
/// </summary>
public class SESv2Wrapper
{

    private readonly IAmazonSimpleEmailServiceV2 _sesClient;

    /// <summary>
    /// Constructor for the SESv2Wrapper.
    /// </summary>
    /// <param name="sesClient">The injected SES v2 client.</param>
    public SESv2Wrapper(IAmazonSimpleEmailServiceV2 sesClient)
    {
        _sesClient = sesClient;
    }

    /// <summary>
    /// Creates a contact and adds it to the specified contact list.
    /// </summary>
    /// <param name="emailAddress">The email address of the contact.</param>
    /// <param name="contactListName">The name of the contact list.</param>
    /// <returns>The response from the CreateContact operation.</returns>
    public async Task<bool> CreateContactAsync(string emailAddress, string
 contactListName)
    {
        var request = new CreateContactRequest
        {
```

```
            EmailAddress = emailAddress,
            ContactListName = contactListName
        };

        try
        {
            var response = await _sesClient.CreateContactAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (AlreadyExistsException ex)
        {
            Console.WriteLine($"Contact with email address {emailAddress} already
exists in the contact list {contactListName}.");
            Console.WriteLine(ex.Message);
            return true;
        }
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The contact list {contactListName} does not
exist.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating the contact:
{ex.Message}");
        }
        return false;
    }

    /// <summary>
    /// Creates a contact list with the specified name.
    /// </summary>
    /// <param name="contactListName">The name of the contact list.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> CreateContactListAsync(string contactListName)
    {
        var request = new CreateContactListRequest
```

```
        {
            ContactListName = contactListName
        };

        try
        {
            var response = await _sesClient.CreateContactListAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (AlreadyExistsException ex)
        {
            Console.WriteLine($"Contact list with name {contactListName} already
exists.");
            Console.WriteLine(ex.Message);
            return true;
        }
        catch (LimitExceededException ex)
        {
            Console.WriteLine("The limit for contact lists has been exceeded.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating the contact list:
{ex.Message}");
        }
        return false;
    }

    /// <summary>
    /// Creates an email identity (email address or domain) and starts the
verification process.
    /// </summary>
    /// <param name="emailIdentity">The email address or domain to create and
verify.</param>
    /// <returns>The response from the CreateEmailIdentity operation.</returns>
    public async Task<CreateEmailIdentityResponse> CreateEmailIdentityAsync(string
emailIdentity)
```

```
    {
        var request = new CreateEmailIdentityRequest
        {
            EmailIdentity = emailIdentity
        };

        try
        {
            var response = await _sesClient.CreateEmailIdentityAsync(request);
            return response;
        }
        catch (AlreadyExistsException ex)
        {
            Console.WriteLine($"Email identity {emailIdentity} already exists.");
            Console.WriteLine(ex.Message);
            throw;
        }
        catch (ConcurrentModificationException ex)
        {
            Console.WriteLine($"The email identity {emailIdentity} is being modified
by another operation or thread.");
            Console.WriteLine(ex.Message);
            throw;
        }
        catch (LimitExceededException ex)
        {
            Console.WriteLine("The limit for email identities has been exceeded.");
            Console.WriteLine(ex.Message);
            throw;
        }
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The email identity {emailIdentity} does not
exist.");
            Console.WriteLine(ex.Message);
            throw;
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
            throw;
        }
```

```
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating the email identity:
{ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Creates an email template with the specified content.
    /// </summary>
    /// <param name="templateName">The name of the email template.</param>
    /// <param name="subject">The subject of the email template.</param>
    /// <param name="htmlContent">The HTML content of the email template.</param>
    /// <param name="textContent">The text content of the email template.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> CreateEmailTemplateAsync(string templateName, string
subject, string htmlContent, string textContent)
    {
        var request = new CreateEmailTemplateRequest
        {
            TemplateName = templateName,
            TemplateContent = new EmailTemplateContent
            {
                Subject = subject,
                Html = htmlContent,
                Text = textContent
            }
        };

        try
        {
            var response = await _sesClient.CreateEmailTemplateAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (AlreadyExistsException ex)
        {
            Console.WriteLine($"Email template with name {templateName} already
exists.");
            Console.WriteLine(ex.Message);
        }
        catch (LimitExceededException ex)
        {
            Console.WriteLine("The limit for email templates has been exceeded.");
```

```
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
 later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating the email template:
 {ex.Message}");
        }

        return false;
    }

    /// <summary>
    /// Deletes a contact list and all contacts within it.
    /// </summary>
    /// <param name="contactListName">The name of the contact list to delete.</
param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteContactListAsync(string contactListName)
    {
        var request = new DeleteContactListRequest
        {
            ContactListName = contactListName
        };

        try
        {
            var response = await _sesClient.DeleteContactListAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (ConcurrentModificationException ex)
        {
            Console.WriteLine($"The contact list {contactListName} is being modified
 by another operation or thread.");
            Console.WriteLine(ex.Message);
        }
        catch (NotFoundException ex)
        {
```

```
            Console.WriteLine($"The contact list {contactListName} does not
exist.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting the contact list:
{ex.Message}");
        }

        return false;
    }

    /// <summary>
    /// Deletes an email identity (email address or domain).
    /// </summary>
    /// <param name="emailIdentity">The email address or domain to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteEmailIdentityAsync(string emailIdentity)
    {
        var request = new DeleteEmailIdentityRequest
        {
            EmailIdentity = emailIdentity
        };

        try
        {
            var response = await _sesClient.DeleteEmailIdentityAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (ConcurrentModificationException ex)
        {
            Console.WriteLine($"The email identity {emailIdentity} is being modified
by another operation or thread.");
            Console.WriteLine(ex.Message);
        }
        catch (NotFoundException ex)
        {
```

```
            Console.WriteLine($"The email identity {emailIdentity} does not
exist.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting the email identity:
{ex.Message}");
        }

        return false;
    }

    /// <summary>
    /// Deletes an email template.
    /// </summary>
    /// <param name="templateName">The name of the email template to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteEmailTemplateAsync(string templateName)
    {
        var request = new DeleteEmailTemplateRequest
        {
            TemplateName = templateName
        };

        try
        {
            var response = await _sesClient.DeleteEmailTemplateAsync(request);
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The email template {templateName} does not exist.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
```

```
            Console.WriteLine("Too many requests were made. Please try again
 later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting the email template:
 {ex.Message}");
        }

        return false;
    }

    /// <summary>
    /// Lists the contacts in the specified contact list.
    /// </summary>
    /// <param name="contactListName">The name of the contact list.</param>
    /// <returns>The list of contacts response from the ListContacts operation.</
returns>
    public async Task<List<Contact>> ListContactsAsync(string contactListName)
    {
        var request = new ListContactsRequest
        {
            ContactListName = contactListName
        };

        try
        {
            var response = await _sesClient.ListContactsAsync(request);
            return response.Contacts;
        }
        catch (NotFoundException ex)
        {
            Console.WriteLine($"The contact list {contactListName} does not
 exist.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
 later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
```

```
        {
            Console.WriteLine($"An error occurred while listing the contacts:
 {ex.Message}");
        }

        return new List<Contact>();
    }

    /// <summary>
    /// Sends an email with the specified content and options.
    /// </summary>
    /// <param name="fromEmailAddress">The email address to send the email from.</
param>
    /// <param name="toEmailAddresses">The email addresses to send the email to.</
param>
    /// <param name="subject">The subject of the email.</param>
    /// <param name="htmlContent">The HTML content of the email.</param>
    /// <param name="textContent">The text content of the email.</param>
    /// <param name="templateName">The name of the email template to use
 (optional).</param>
    /// <param name="templateData">The data to replace placeholders in the email
 template (optional).</param>
    /// <param name="contactListName">The name of the contact list for unsubscribe
 functionality (optional).</param>
    /// <returns>The MessageId response from the SendEmail operation.</returns>
    public async Task<string> SendEmailAsync(string fromEmailAddress, List<string>
 toEmailAddresses, string? subject,
        string? htmlContent, string? textContent, string? templateName = null,
 string? templateData = null, string? contactListName = null)
    {
        var request = new SendEmailRequest
        {
            FromEmailAddress = fromEmailAddress
        };

        if (toEmailAddresses.Any())
        {
            request.Destination = new Destination { ToAddresses =
 toEmailAddresses };
        }

        if (!string.IsNullOrEmpty(templateName))
        {
            request.Content = new EmailContent()
```

```
            {
                Template = new Template
                {
                    TemplateName = templateName,
                    TemplateData = templateData
                }
            };
        }
        else
        {
            request.Content = new EmailContent
            {
                Simple = new Message
                {
                    Subject = new Content { Data = subject },
                    Body = new Body
                    {
                        Html = new Content { Data = htmlContent },
                        Text = new Content { Data = textContent }
                    }
                }
            };
        }

        if (!string.IsNullOrEmpty(contactListName))
        {
            request.ListManagementOptions = new ListManagementOptions
            {
                ContactListName = contactListName
            };
        }

        try
        {
            var response = await _sesClient.SendEmailAsync(request);
            return response.MessageId;
        }
        catch (AccountSuspendedException ex)
        {
            Console.WriteLine("The account's ability to send email has been
permanently restricted.");
            Console.WriteLine(ex.Message);
        }
        catch (MailFromDomainNotVerifiedException ex)
```

```
        {
            Console.WriteLine("The sending domain is not verified.");
            Console.WriteLine(ex.Message);
        }
        catch (MessageRejectedException ex)
        {
            Console.WriteLine("The message content is invalid.");
            Console.WriteLine(ex.Message);
        }
        catch (SendingPausedException ex)
        {
            Console.WriteLine("The account's ability to send email is currently
 paused.");
            Console.WriteLine(ex.Message);
        }
        catch (TooManyRequestsException ex)
        {
            Console.WriteLine("Too many requests were made. Please try again
 later.");
            Console.WriteLine(ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while sending the email:
 {ex.Message}");
        }

        return string.Empty;
    }
}
```

- 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。
  - CreateContact
  - CreateContactList
  - CreateEmailIdentity
  - CreateEmailTemplate
  - DeleteContactList
  - DeleteEmailIdentity

- DeleteEmailTemplate
- ListContacts
- SendEmailシンプル
- SendEmailテンプレート

を使用した Amazon SNS の例 AWS SDK for .NET

次のコード例は、Amazon SNS AWS SDK for .NET で 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。 Amazon SNS

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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。このリンクには GitHub、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が記載されています。

開始方法

Hello Amazon SNS

以下のコード例は、Amazon SNS の使用を開始する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;
```

```
namespace SNSActions;

public static class HelloSNS
{
    static async Task Main(string[] args)
    {
        var snsClient = new AmazonSimpleNotificationServiceClient();

        Console.WriteLine($"Hello Amazon SNS! Following are some of your topics:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get a list of topics.
        var response = await snsClient.ListTopicsAsync(
            new ListTopicsRequest());

        foreach (var topic in response.Topics)
        {
            Console.WriteLine($"\tTopic ARN: {topic.TopicArn}");
            Console.WriteLine();
        }
    }
}
```

- API の詳細については、「 API リファレンスListTopics」の「」を参照してください。 AWS SDK for .NET

トピック

- アクション
- シナリオ
- サーバーレスサンプル

アクション

`CheckIfPhoneNumberIsOptedOut`

次の例は、CheckIfPhoneNumberIsOptedOut を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

    /// <summary>
    /// This example shows how to use the Amazon Simple Notification Service
    /// (Amazon SNS) to check whether a phone number has been opted out.
    /// </summary>
    public class IsPhoneNumOptedOut
    {
        public static async Task Main()
        {
            string phoneNumber = "+15551112222";

            IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

            await CheckIfOptedOutAsync(client, phoneNumber);
        }

        /// <summary>
        /// Checks to see if the supplied phone number has been opted out.
        /// </summary>
        /// <param name="client">The initialized Amazon SNS Client object used
        /// to check if the phone number has been opted out.</param>
        /// <param name="phoneNumber">A string representing the phone number
        /// to check.</param>
        public static async Task
CheckIfOptedOutAsync(IAmazonSimpleNotificationService client, string phoneNumber)
        {
            var request = new CheckIfPhoneNumberIsOptedOutRequest
            {
                PhoneNumber = phoneNumber,
```

```
            };

            try
            {
                var response = await
client.CheckIfPhoneNumberIsOptedOutAsync(request);

                if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
                {
                    string optOutStatus = response.IsOptedOut ? "opted out" : "not
opted out.";
                    Console.WriteLine($"The phone number: {phoneNumber} is
{optOutStatus}");
                }
            }
            catch (AuthorizationErrorException ex)
            {
                Console.WriteLine($"{ex.Message}");
            }
        }
    }
```

- API の詳細については、「 API リファレンスCheckIfPhoneNumberIsOptedOut」の「」を参照してください。 AWS SDK for .NET

`CreateTopic`

次の例は、`CreateTopic` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

トピックを作成して、個別の名前を付けます。

```
    using System;
```

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

   /// <summary>
   /// This example shows how to use Amazon Simple Notification Service
   /// (Amazon SNS) to add a new Amazon SNS topic.
   /// </summary>
   public class CreateSNSTopic
   {
       public static async Task Main()
       {
           string topicName = "ExampleSNSTopic";

           IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

           var topicArn = await CreateSNSTopicAsync(client, topicName);
           Console.WriteLine($"New topic ARN: {topicArn}");
       }

       /// <summary>
       /// Creates a new SNS topic using the supplied topic name.
       /// </summary>
       /// <param name="client">The initialized SNS client object used to
       /// create the new topic.</param>
       /// <param name="topicName">A string representing the topic name.</param>
       /// <returns>The Amazon Resource Name (ARN) of the created topic.</returns>
       public static async Task<string>
CreateSNSTopicAsync(IAmazonSimpleNotificationService client, string topicName)
       {
           var request = new CreateTopicRequest
           {
               Name = topicName,
           };

           var response = await client.CreateTopicAsync(request);

           return response.TopicArn;
       }
   }
```

名前と特定の FIFO および重複除外属性を使用して新しいトピックを作成します。

```
    /// <summary>
    /// Create a new topic with a name and specific FIFO and de-duplication
 attributes.
    /// </summary>
    /// <param name="topicName">The name for the topic.</param>
    /// <param name="useFifoTopic">True to use a FIFO topic.</param>
    /// <param name="useContentBasedDeduplication">True to use content-based de-
duplication.</param>
    /// <returns>The ARN of the new topic.</returns>
    public async Task<string> CreateTopicWithName(string topicName, bool
 useFifoTopic, bool useContentBasedDeduplication)
    {
        var createTopicRequest = new CreateTopicRequest()
        {
            Name = topicName,
        };

        if (useFifoTopic)
        {
            // Update the name if it is not correct for a FIFO topic.
            if (!topicName.EndsWith(".fifo"))
            {
                createTopicRequest.Name = topicName + ".fifo";
            }

            // Add the attributes from the method parameters.
            createTopicRequest.Attributes = new Dictionary<string, string>
            {
                { "FifoTopic", "true" }
            };
            if (useContentBasedDeduplication)
            {
                createTopicRequest.Attributes.Add("ContentBasedDeduplication",
 "true");
            }
        }

        var createResponse = await
 _amazonSNSClient.CreateTopicAsync(createTopicRequest);
        return createResponse.TopicArn;
    }
```

- API の詳細については、「 API リファレンスCreateTopic」の「」を参照してください。 AWS SDK for .NET

`DeleteTopic`

次の例は、DeleteTopic を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

トピック ARN でトピックを削除します。

```
/// <summary>
/// Delete a topic by its topic ARN.
/// </summary>
/// <param name="topicArn">The ARN of the topic.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteTopicByArn(string topicArn)
{
    var deleteResponse = await _amazonSNSClient.DeleteTopicAsync(
        new DeleteTopicRequest()
        {
            TopicArn = topicArn
        });
    return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
}
```

- API の詳細については、「 API リファレンスDeleteTopic」の「」を参照してください。 AWS SDK for .NET

`GetTopicAttributes`

次の例は、GetTopicAttributes を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon.SimpleNotificationService;

    /// <summary>
    /// This example shows how to retrieve the attributes of an Amazon Simple
    /// Notification Service (Amazon SNS) topic.
    /// </summary>
    public class GetTopicAttributes
    {
        public static async Task Main()
        {
            string topicArn = "arn:aws:sns:us-west-2:000000000000:ExampleSNSTopic";
            IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

            var attributes = await GetTopicAttributesAsync(client, topicArn);
            DisplayTopicAttributes(attributes);
        }

        /// <summary>
        /// Given the ARN of the Amazon SNS topic, this method retrieves the topic
        /// attributes.
        /// </summary>
        /// <param name="client">The initialized Amazon SNS client object used
        /// to retrieve the attributes for the Amazon SNS topic.</param>
        /// <param name="topicArn">The ARN of the topic for which to retrieve
        /// the attributes.</param>
        /// <returns>A Dictionary of topic attributes.</returns>
        public static async Task<Dictionary<string, string>>
GetTopicAttributesAsync(
            IAmazonSimpleNotificationService client,
            string topicArn)
```

```
        {
            var response = await client.GetTopicAttributesAsync(topicArn);

            return response.Attributes;
        }

        /// <summary>
        /// This method displays the attributes for an Amazon SNS topic.
        /// </summary>
        /// <param name="topicAttributes">A Dictionary containing the
        /// attributes for an Amazon SNS topic.</param>
        public static void DisplayTopicAttributes(Dictionary<string, string>
topicAttributes)
        {
            foreach (KeyValuePair<string, string> entry in topicAttributes)
            {
                Console.WriteLine($"{entry.Key}: {entry.Value}\n");
            }
        }
    }
```

- API の詳細については、「 API リファレンスGetTopicAttributes」の「」を参照してください。AWS SDK for .NET

`ListSubscriptions`

次の例は、`ListSubscriptions` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
```

```
    using Amazon.SimpleNotificationService;
    using Amazon.SimpleNotificationService.Model;

    /// <summary>
    /// This example will retrieve a list of the existing Amazon Simple
    /// Notification Service (Amazon SNS) subscriptions.
    /// </summary>
    public class ListSubscriptions
    {
        public static async Task Main()
        {
            IAmazonSimpleNotificationService client = new
 AmazonSimpleNotificationServiceClient();

            Console.WriteLine("Enter a topic ARN to list subscriptions for a
 specific topic, " +
                              "or press Enter to list subscriptions for all
 topics.");
            var topicArn = Console.ReadLine();
            Console.WriteLine();

            var subscriptions = await GetSubscriptionsListAsync(client, topicArn);

            DisplaySubscriptionList(subscriptions);
        }

        /// <summary>
        /// Gets a list of the existing Amazon SNS subscriptions, optionally by
 specifying a topic ARN.
        /// </summary>
        /// <param name="client">The initialized Amazon SNS client object used
        /// to obtain the list of subscriptions.</param>
        /// <param name="topicArn">The optional ARN of a specific topic. Defaults to
 null.</param>
        /// <returns>A list containing information about each subscription.</
returns>
        public static async Task<List<Subscription>>
 GetSubscriptionsListAsync(IAmazonSimpleNotificationService client, string topicArn
 = null)
        {
            var results = new List<Subscription>();

            if (!string.IsNullOrEmpty(topicArn))
            {
```

```
                var paginateByTopic = client.Paginators.ListSubscriptionsByTopic(
                    new ListSubscriptionsByTopicRequest()
                    {
                        TopicArn = topicArn,
                    });

                // Get the entire list using the paginator.
                await foreach (var subscription in paginateByTopic.Subscriptions)
                {
                    results.Add(subscription);
                }
            }
            else
            {
                var paginateAllSubscriptions =
client.Paginators.ListSubscriptions(new ListSubscriptionsRequest());

                // Get the entire list using the paginator.
                await foreach (var subscription in
paginateAllSubscriptions.Subscriptions)
                {
                    results.Add(subscription);
                }
            }

            return results;
        }

        /// <summary>
        /// Display a list of Amazon SNS subscription information.
        /// </summary>
        /// <param name="subscriptionList">A list containing details for existing
        /// Amazon SNS subscriptions.</param>
        public static void DisplaySubscriptionList(List<Subscription>
subscriptionList)
        {
            foreach (var subscription in subscriptionList)
            {
                Console.WriteLine($"Owner: {subscription.Owner}");
                Console.WriteLine($"Subscription ARN:
{subscription.SubscriptionArn}");
                Console.WriteLine($"Topic ARN: {subscription.TopicArn}");
                Console.WriteLine($"Endpoint: {subscription.Endpoint}");
                Console.WriteLine($"Protocol: {subscription.Protocol}");
```

```
                Console.WriteLine();
            }
        }
    }
```

- API の詳細については、「 API リファレンスListSubscriptions」の「」を参照してください。 AWS SDK for .NET

`ListTopics`

次の例は、`ListTopics` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon.SimpleNotificationService;
    using Amazon.SimpleNotificationService.Model;

    /// <summary>
    /// Lists the Amazon Simple Notification Service (Amazon SNS)
    /// topics for the current account.
    /// </summary>
    public class ListSNSTopics
    {
        public static async Task Main()
        {
            IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

            await GetTopicListAsync(client);
        }
```

```
        /// <summary>
        /// Retrieves the list of Amazon SNS topics in groups of up to 100
        /// topics.
        /// </summary>
        /// <param name="client">The initialized Amazon SNS client object used
        /// to retrieve the list of topics.</param>
        public static async Task GetTopicListAsync(IAmazonSimpleNotificationService
client)
        {
            // If there are more than 100 Amazon SNS topics, the call to
            // ListTopicsAsync will return a value to pass to the
            // method to retrieve the next 100 (or less) topics.
            string nextToken = string.Empty;

            do
            {
                var response = await client.ListTopicsAsync(nextToken);
                DisplayTopicsList(response.Topics);
                nextToken = response.NextToken;
            }
            while (!string.IsNullOrEmpty(nextToken));
        }

        /// <summary>
        /// Displays the list of Amazon SNS Topic ARNs.
        /// </summary>
        /// <param name="topicList">The list of Topic ARNs.</param>
        public static void DisplayTopicsList(List<Topic> topicList)
        {
            foreach (var topic in topicList)
            {
                Console.WriteLine($"{topic.TopicArn}");
            }
        }
    }
```

- API の詳細については、「 API リファレンスListTopics」の「」を参照してください。 AWS SDK for .NET

Publish

次の例は、Publish を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

トピックへのメッセージの発行

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

    /// <summary>
    /// This example publishes a message to an Amazon Simple Notification
    /// Service (Amazon SNS) topic.
    /// </summary>
    public class PublishToSNSTopic
    {
        public static async Task Main()
        {
            string topicArn = "arn:aws:sns:us-east-2:000000000000:ExampleSNSTopic";
            string messageText = "This is an example message to publish to the
ExampleSNSTopic.";

            IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

            await PublishToTopicAsync(client, topicArn, messageText);
        }

        /// <summary>
        /// Publishes a message to an Amazon SNS topic.
        /// </summary>
        /// <param name="client">The initialized client object used to publish
        /// to the Amazon SNS topic.</param>
        /// <param name="topicArn">The ARN of the topic.</param>
```

```
        /// <param name="messageText">The text of the message.</param>
        public static async Task PublishToTopicAsync(
            IAmazonSimpleNotificationService client,
            string topicArn,
            string messageText)
        {
            var request = new PublishRequest
            {
                TopicArn = topicArn,
                Message = messageText,
            };

            var response = await client.PublishAsync(request);

            Console.WriteLine($"Successfully published message ID:
{response.MessageId}");
        }
    }
```

グループ、重複、属性のオプションを指定してメッセージをトピックに発行します。

```
    /// <summary>
    /// Publish messages using user settings.
    /// </summary>
    /// <returns>Async task.</returns>
    public static async Task PublishMessages()
    {
        Console.WriteLine("Now we can publish messages.");

        var keepSendingMessages = true;
        string? deduplicationId = null;
        string? toneAttribute = null;
        while (keepSendingMessages)
        {
            Console.WriteLine();
            var message = GetUserResponse("Enter a message to publish.", "This is a
sample message");

            if (_useFifoTopic)
            {
```

```
                Console.WriteLine("Because you are using a FIFO topic, you must set
a message group ID." +
                                  "\r\nAll messages within the same group will be
received in the order " +
                                  "they were published.");

                Console.WriteLine();
                var messageGroupId = GetUserResponse("Enter a message group ID for
this message:", "1");

                if (!_useContentBasedDeduplication)
                {
                    Console.WriteLine("Because you are not using content-based
deduplication, " +
                                      "you must enter a deduplication ID.");

                    Console.WriteLine("Enter a deduplication ID for this message.");
                    deduplicationId = GetUserResponse("Enter a deduplication ID for
this message.", "1");
                }

                if (GetYesNoResponse("Add an attribute to this message?"))
                {
                    Console.WriteLine("Enter a number for an attribute.");
                    for (int i = 0; i < _tones.Length; i++)
                    {
                        Console.WriteLine($"\t{i + 1}. {_tones[i]}");
                    }

                    var selection = GetUserResponse("", "1");
                    int.TryParse(selection, out var selectionNumber);

                    if (selectionNumber > 0 && selectionNumber < _tones.Length)
                    {
                        toneAttribute = _tones[selectionNumber - 1];
                    }
                }

                var messageID = await SnsWrapper.PublishToTopicWithAttribute(
                    _topicArn, message, "tone", toneAttribute, deduplicationId,
messageGroupId);

                Console.WriteLine($"Message published with id {messageID}.");
            }
```

```
            keepSendingMessages = GetYesNoResponse("Send another message?", false);
        }
    }
```

ユーザーの選択を発行アクションに適用します。

```
    /// <summary>
    /// Publish a message to a topic with an attribute and optional deduplication
 and group IDs.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="message">The message to publish.</param>
    /// <param name="attributeName">The optional attribute for the message.</param>
    /// <param name="attributeValue">The optional attribute value for the message.</
param>
    /// <param name="deduplicationId">The optional deduplication ID for the
 message.</param>
    /// <param name="groupId">The optional group ID for the message.</param>
    /// <returns>The ID of the message published.</returns>
    public async Task<string> PublishToTopicWithAttribute(
        string topicArn,
        string message,
        string? attributeName = null,
        string? attributeValue = null,
        string? deduplicationId = null,
        string? groupId = null)
    {
        var publishRequest = new PublishRequest()
        {
            TopicArn = topicArn,
            Message = message,
            MessageDeduplicationId = deduplicationId,
            MessageGroupId = groupId
        };

        if (attributeValue != null)
        {
            // Add the string attribute if it exists.
            publishRequest.MessageAttributes =
                new Dictionary<string, MessageAttributeValue>
                {
```

```
                    { attributeName!, new MessageAttributeValue() { StringValue =
attributeValue, DataType = "String"} }
                };
        }

        var publishResponse = await _amazonSNSClient.PublishAsync(publishRequest);
        return publishResponse.MessageId;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「Publish」を参照してください。

Subscribe

次の例は、Subscribe を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

E メールアドレスをトピックにサブスクライブします。

```
        /// <summary>
        /// Creates a new subscription to a topic.
        /// </summary>
        /// <param name="client">The initialized Amazon SNS client object, used
        /// to create an Amazon SNS subscription.</param>
        /// <param name="topicArn">The ARN of the topic to subscribe to.</param>
        /// <returns>A SubscribeResponse object which includes the subscription
        /// ARN for the new subscription.</returns>
        public static async Task<SubscribeResponse> TopicSubscribeAsync(
            IAmazonSimpleNotificationService client,
            string topicArn)
        {
            SubscribeRequest request = new SubscribeRequest()
            {
```

```
                TopicArn = topicArn,
                ReturnSubscriptionArn = true,
                Protocol = "email",
                Endpoint = "recipient@example.com",
            };

            var response = await client.SubscribeAsync(request);

            return response;
        }
```

オプションのフィルターでトピックにキューをサブスクライブします。

```
    /// <summary>
    /// Subscribe a queue to a topic with optional filters.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="useFifoTopic">The optional filtering policy for the
subscription.</param>
    /// <param name="queueArn">The ARN of the queue.</param>
    /// <returns>The ARN of the new subscription.</returns>
    public async Task<string> SubscribeTopicWithFilter(string topicArn, string?
filterPolicy, string queueArn)
    {
        var subscribeRequest = new SubscribeRequest()
        {
            TopicArn = topicArn,
            Protocol = "sqs",
            Endpoint = queueArn
        };

        if (!string.IsNullOrEmpty(filterPolicy))
        {
            subscribeRequest.Attributes = new Dictionary<string, string>
{ { "FilterPolicy", filterPolicy } };
        }

        var subscribeResponse = await
_amazonSNSClient.SubscribeAsync(subscribeRequest);
        return subscribeResponse.SubscriptionArn;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「Subscribe」を参照してください。

`Unsubscribe`

次の例は、`Unsubscribe` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

サブスクリプション ARN でトピックからサブスクライブを解除します。

```
/// <summary>
/// Unsubscribe from a topic by a subscription ARN.
/// </summary>
/// <param name="subscriptionArn">The ARN of the subscription.</param>
/// <returns>True if successful.</returns>
public async Task<bool> UnsubscribeByArn(string subscriptionArn)
{
    var unsubscribeResponse = await _amazonSNSClient.UnsubscribeAsync(
        new UnsubscribeRequest()
        {
            SubscriptionArn = subscriptionArn
        });
    return unsubscribeResponse.HttpStatusCode == HttpStatusCode.OK;
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「Unsubscribe」を参照してください。

シナリオ

SMS テキストメッセージを発行する

次のコードサンプルは、Amazon SNS を使用して SMS メッセージを発行する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
namespace SNSMessageExample
{
    using System;
    using System.Threading.Tasks;
    using Amazon;
    using Amazon.SimpleNotificationService;
    using Amazon.SimpleNotificationService.Model;

    public class SNSMessage
    {
        private AmazonSimpleNotificationServiceClient snsClient;

        /// <summary>
        /// Initializes a new instance of the <see cref="SNSMessage"/> class.
        /// Constructs a new SNSMessage object initializing the Amazon Simple
        /// Notification Service (Amazon SNS) client using the supplied
        /// Region endpoint.
        /// </summary>
        /// <param name="regionEndpoint">The Amazon Region endpoint to use in
        /// sending test messages with this object.</param>
        public SNSMessage(RegionEndpoint regionEndpoint)
        {
            snsClient = new AmazonSimpleNotificationServiceClient(regionEndpoint);
        }

        /// <summary>
        /// Sends the SMS message passed in the text parameter to the phone number
        /// in phoneNum.
        /// </summary>
```

```
        /// <param name="phoneNum">The ten-digit phone number to which the text
        /// message will be sent.</param>
        /// <param name="text">The text of the message to send.</param>
        /// <returns>Async task.</returns>
        public async Task SendTextMessageAsync(string phoneNum, string text)
        {
            if (string.IsNullOrEmpty(phoneNum) || string.IsNullOrEmpty(text))
            {
                return;
            }

            // Now actually send the message.
            var request = new PublishRequest
            {
                Message = text,
                PhoneNumber = phoneNum,
            };

            try
            {
                var response = await snsClient.PublishAsync(request);
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Error sending message: {ex}");
            }
        }
    }
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「Publish」を参照してください。

メッセージをキューに発行する

次のコードサンプルは、以下の操作方法を示しています。

- トピック (FIFO または非 FIFO) を作成します。
- フィルターを適用するオプションを使用して、複数のキューをトピックにサブスクライブします。
- メッセージをトピックに発行します。

- キューをポーリングして受信メッセージを確認します。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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のシナリオを実行します。

```
/// <summary>
/// Console application to run a workflow scenario for topics and queues.
/// </summary>
public static class TopicsAndQueues
{
    private static bool _useFifoTopic = false;
    private static bool _useContentBasedDeduplication = false;
    private static string _topicName = null!;
    private static string _topicArn = null!;

    private static readonly int _queueCount = 2;
    private static readonly string[] _queueUrls = new string[_queueCount];
    private static readonly string[] _subscriptionArns = new string[_queueCount];
    private static readonly string[] _tones = { "cheerful", "funny", "serious",
 "sincere" };
    public static SNSWrapper SnsWrapper { get; set; } = null!;
    public static SQSWrapper SqsWrapper { get; set; } = null!;
    public static bool UseConsole { get; set; } = true;
    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon EventBridge.
        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<IAmazonSQS>()
                    .AddAWSService<IAmazonSimpleNotificationService>()
```

```
                    .AddTransient<SNSWrapper>()
                    .AddTransient<SQSWrapper>()
            )
            .Build();

        ServicesSetup(host);
        PrintDescription();

        await RunScenario();

    }

    /// <summary>
    /// Populate the services for use within the console application.
    /// </summary>
    /// <param name="host">The services host.</param>
    private static void ServicesSetup(IHost host)
    {
        SnsWrapper = host.Services.GetRequiredService<SNSWrapper>();
        SqsWrapper = host.Services.GetRequiredService<SQSWrapper>();
    }

    /// <summary>
    /// Run the scenario for working with topics and queues.
    /// </summary>
    /// <returns>True if successful.</returns>
    public static async Task<bool> RunScenario()
    {
        try
        {
            await SetupTopic();

            await SetupQueues();

            await PublishMessages();

            foreach (var queueUrl in _queueUrls)
            {
                var messages = await PollForMessages(queueUrl);
                if (messages.Any())
                {
                    await DeleteMessages(queueUrl, messages);
                }
            }
```

```
            await CleanupResources();

            Console.WriteLine("Messaging with topics and queues workflow is
complete.");
            return true;
        }
        catch (Exception ex)
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
            await CleanupResources();
            Console.WriteLine(new string('-', 80));
            return false;
        }
    }

    /// <summary>
    /// Print a description for the tasks in the workflow.
    /// </summary>
    /// <returns>Async task.</returns>
    private static void PrintDescription()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Welcome to messaging with topics and queues.");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"In this workflow, you will create an SNS topic and
subscribe {_queueCount} SQS queues to the topic." +
                          $"\r\nYou can select from several options for configuring
the topic and the subscriptions for the 2 queues." +
                          $"\r\nYou can then post to the topic and see the results
in the queues.\r\n");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Set up the SNS topic to be used with the queues.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task<string> SetupTopic()
    {
        Console.WriteLine(new string('-', 80));
```

```
        Console.WriteLine($"SNS topics can be configured as FIFO (First-In-First-
Out)." +
                          $"\r\nFIFO topics deliver messages in order and support
 deduplication and message filtering." +
                          $"\r\nYou can then post to the topic and see the results
 in the queues.\r\n");

        _useFifoTopic = GetYesNoResponse("Would you like to work with FIFO
 topics?");

        if (_useFifoTopic)
        {
            Console.WriteLine(new string('-', 80));
            _topicName = GetUserResponse("Enter a name for your SNS topic: ",
 "example-topic");
            Console.WriteLine(
                "Because you have selected a FIFO topic, '.fifo' must be appended to
 the topic name.\r\n");

            Console.WriteLine(new string('-', 80));
            Console.WriteLine($"Because you have chosen a FIFO topic, deduplication
 is supported." +
                              $"\r\nDeduplication IDs are either set in the message
 or automatically generated " +
                              $"\r\nfrom content using a hash function.\r\n" +
                              $"\r\nIf a message is successfully published to an SNS
 FIFO topic, any message " +
                              $"\r\npublished and determined to have the same
 deduplication ID, " +
                              $"\r\nwithin the five-minute deduplication interval,
 is accepted but not delivered.\r\n" +
                              $"\r\nFor more information about deduplication, " +
                              $"\r\nsee https://docs.aws.amazon.com/sns/latest/dg/
fifo-message-dedup.html.");

            _useContentBasedDeduplication = GetYesNoResponse("Use content-based
 deduplication instead of entering a deduplication ID?");
            Console.WriteLine(new string('-', 80));
        }

        _topicArn = await SnsWrapper.CreateTopicWithName(_topicName, _useFifoTopic,
 _useContentBasedDeduplication);

        Console.WriteLine($"Your new topic with the name {_topicName}" +
```

```
                        $"\r\nand Amazon Resource Name (ARN) {_topicArn}" +
                        $"\r\nhas been created.\r\n");

        Console.WriteLine(new string('-', 80));
        return _topicArn;
    }

    /// <summary>
    /// Set up the queues.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task SetupQueues()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Now you will create {_queueCount} Amazon Simple Queue
Service (Amazon SQS) queues to subscribe to the topic.");

        // Repeat this section for each queue.
        for (int i = 0; i < _queueCount; i++)
        {
            var queueName = GetUserResponse("Enter a name for an Amazon SQS queue:
", $"example-queue-{i}");
            if (_useFifoTopic)
            {
                // Only explain this once.
                if (i == 0)
                {
                    Console.WriteLine(
                        "Because you have selected a FIFO topic, '.fifo' must be
appended to the queue name.");
                }

                var queueUrl = await SqsWrapper.CreateQueueWithName(queueName,
_useFifoTopic);

                _queueUrls[i] = queueUrl;

                Console.WriteLine($"Your new queue with the name {queueName}" +
                                  $"\r\nand queue URL {queueUrl}" +
                                  $"\r\nhas been created.\r\n");

                if (i == 0)
                {
                    Console.WriteLine(
```

```
                        $"The queue URL is used to retrieve the queue ARN,\r\n" +
                        $"which is used to create a subscription.");
                    Console.WriteLine(new string('-', 80));
                }

                var queueArn = await SqsWrapper.GetQueueArnByUrl(queueUrl);

                if (i == 0)
                {
                    Console.WriteLine(
                        $"An AWS Identity and Access Management (IAM) policy must be
attached to an SQS queue, enabling it to receive\r\n" +
                        $"messages from an SNS topic");
                }

                await SqsWrapper.SetQueuePolicyForTopic(queueArn, _topicArn,
queueUrl);

                await SetupFilters(i, queueArn, queueName);
            }
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Set up filters with user options for a queue.
    /// </summary>
    /// <param name="queueCount">The number of this queue.</param>
    /// <param name="queueArn">The ARN of the queue.</param>
    /// <param name="queueName">The name of the queue.</param>
    /// <returns>Async Task.</returns>
    public static async Task SetupFilters(int queueCount, string queueArn, string
queueName)
    {
        if (_useFifoTopic)
        {
            Console.WriteLine(new string('-', 80));
            // Only explain this once.
            if (queueCount == 0)
            {
                Console.WriteLine(
                    "Subscriptions to a FIFO topic can have filters." +
```

```
                    "If you add a filter to this subscription, then only the
 filtered messages " +
                    "will be received in the queue.");

                Console.WriteLine(
                    "For information about message filtering, " +
                    "see https://docs.aws.amazon.com/sns/latest/dg/sns-message-
filtering.html");

                Console.WriteLine(
                    "For this example, you can filter messages by a" +
                    "TONE attribute.");
            }

            var useFilter = GetYesNoResponse($"Filter messages for {queueName}'s
 subscription to the topic?");

            string? filterPolicy = null;
            if (useFilter)
            {
                filterPolicy = CreateFilterPolicy();
            }
            var subscriptionArn = await
 SnsWrapper.SubscribeTopicWithFilter(_topicArn, filterPolicy,
                queueArn);
            _subscriptionArns[queueCount] = subscriptionArn;

            Console.WriteLine(
                $"The queue {queueName} has been subscribed to the topic
 {_topicName} " +
                $"with the subscription ARN {subscriptionArn}");
            Console.WriteLine(new string('-', 80));
        }
    }

    /// <summary>
    /// Use user input to create a filter policy for a subscription.
    /// </summary>
    /// <returns>The serialized filter policy.</returns>
    public static string CreateFilterPolicy()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine(
            $"You can filter messages by one or more of the following" +
```

```
            $"TONE attributes.");

        List<string> filterSelections = new List<string>();

        var selectionNumber = 0;
        do
        {
            Console.WriteLine(
                $"Enter a number to add a TONE filter, or enter 0 to stop adding
 filters.");
            for (int i = 0; i < _tones.Length; i++)
            {
                Console.WriteLine($"\t{i + 1}. {_tones[i]}");
            }

            var selection = GetUserResponse("", filterSelections.Any() ? "0" : "1");
            int.TryParse(selection, out selectionNumber);
            if (selectionNumber > 0 &&
filterSelections.Contains(_tones[selectionNumber - 1]))
            {
                filterSelections.Add(_tones[selectionNumber - 1]);
            }
        } while (selectionNumber != 0);

        var filters = new Dictionary<string, List<string>>
        {
            { "tone", filterSelections }
        };
        string filterPolicy = JsonSerializer.Serialize(filters);
        return filterPolicy;
    }

    /// <summary>
    /// Publish messages using user settings.
    /// </summary>
    /// <returns>Async task.</returns>
    public static async Task PublishMessages()
    {
        Console.WriteLine("Now we can publish messages.");

        var keepSendingMessages = true;
        string? deduplicationId = null;
        string? toneAttribute = null;
        while (keepSendingMessages)
```

```
        {
            Console.WriteLine();
            var message = GetUserResponse("Enter a message to publish.", "This is a
sample message");

            if (_useFifoTopic)
            {
                Console.WriteLine("Because you are using a FIFO topic, you must set
a message group ID." +
                                  "\r\nAll messages within the same group will be
received in the order " +
                                  "they were published.");

                Console.WriteLine();
                var messageGroupId = GetUserResponse("Enter a message group ID for
this message:", "1");

                if (!_useContentBasedDeduplication)
                {
                    Console.WriteLine("Because you are not using content-based
deduplication, " +
                                      "you must enter a deduplication ID.");

                    Console.WriteLine("Enter a deduplication ID for this message.");
                    deduplicationId = GetUserResponse("Enter a deduplication ID for
this message.", "1");
                }

                if (GetYesNoResponse("Add an attribute to this message?"))
                {
                    Console.WriteLine("Enter a number for an attribute.");
                    for (int i = 0; i < _tones.Length; i++)
                    {
                        Console.WriteLine($"\t{i + 1}. {_tones[i]}");
                    }

                    var selection = GetUserResponse("", "1");
                    int.TryParse(selection, out var selectionNumber);

                    if (selectionNumber > 0 && selectionNumber < _tones.Length)
                    {
                        toneAttribute = _tones[selectionNumber - 1];
                    }
                }
```

```
                var messageID = await SnsWrapper.PublishToTopicWithAttribute(
                    _topicArn, message, "tone", toneAttribute, deduplicationId,
messageGroupId);

                Console.WriteLine($"Message published with id {messageID}.");
            }

            keepSendingMessages = GetYesNoResponse("Send another message?", false);
        }
    }

    /// <summary>
    /// Poll for the published messages to see the results of the user's choices.
    /// </summary>
    /// <returns>Async task.</returns>
    public static async Task<List<Message>> PollForMessages(string queueUrl)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Now the SQS queue at {queueUrl} will be polled to
retrieve the messages." +
                          "\r\nPress any key to continue.");
        if (UseConsole)
        {
            Console.ReadLine();
        }

        var moreMessages = true;
        var messages = new List<Message>();
        while (moreMessages)
        {
            var newMessages = await SqsWrapper.ReceiveMessagesByUrl(queueUrl, 10);

            moreMessages = newMessages.Any();
            if (moreMessages)
            {
                messages.AddRange(newMessages);
            }
        }

        Console.WriteLine($"{messages.Count} message(s) were received by the queue
at {queueUrl}.");

        foreach (var message in messages)
```

```
        {
            Console.WriteLine("\tMessage:" +
                              $"\n\t{message.Body}");
        }

        Console.WriteLine(new string('-', 80));
        return messages;
    }

    /// <summary>
    /// Delete the message using handles in a batch.
    /// </summary>
    /// <returns>Async task.</returns>
    public static async Task DeleteMessages(string queueUrl, List<Message> messages)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Now we can delete the messages in this queue in a
batch.");
        await SqsWrapper.DeleteMessageBatchByUrl(queueUrl, messages);
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CleanupResources()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Clean up resources.");

        try
        {
            foreach (var queueUrl in _queueUrls)
            {
                if (!string.IsNullOrEmpty(queueUrl))
                {
                    var deleteQueue =
                        GetYesNoResponse($"Delete queue with url {queueUrl}?");
                    if (deleteQueue)
                    {
                        await SqsWrapper.DeleteQueueByUrl(queueUrl);
                    }
                }
```

```
            }

            foreach (var subscriptionArn in _subscriptionArns)
            {
                if (!string.IsNullOrEmpty(subscriptionArn))
                {
                    await SnsWrapper.UnsubscribeByArn(subscriptionArn);
                }
            }

            var deleteTopic = GetYesNoResponse($"Delete topic {_topicName}?");
            if (deleteTopic)
            {
                await SnsWrapper.DeleteTopicByArn(_topicArn);
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Unable to clean up resources. Here's why:
{ex.Message}.");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Helper method to get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</param>
    /// <param name="defaultAnswer">Optional default answer to use.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question, bool defaultAnswer = true)
    {
        if (UseConsole)
        {
            Console.WriteLine(question);
            var ynResponse = Console.ReadLine();
            var response = ynResponse != null &&
                           ynResponse.Equals("y",
                               StringComparison.InvariantCultureIgnoreCase);
            return response;
        }
        // If not using the console, use the default.
        return defaultAnswer;
```

```
    }

    /// <summary>
    /// Helper method to get a string response from the user through the console.
    /// </summary>
    /// <param name="question">The question string to print on the console.</param>
    /// <param name="defaultAnswer">Optional default answer to use.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static string GetUserResponse(string question, string defaultAnswer)
    {
        if (UseConsole)
        {
            var response = "";
            while (string.IsNullOrEmpty(response))
            {
                Console.WriteLine(question);
                response = Console.ReadLine();
            }
            return response;
        }
        // If not using the console, use the default.
        return defaultAnswer;
    }
}
```

Amazon SQS オペレーションをラップするクラスを作成します。

```
/// <summary>
/// Wrapper for Amazon Simple Queue Service (SQS) operations.
/// </summary>
public class SQSWrapper
{
    private readonly IAmazonSQS _amazonSQSClient;

    /// <summary>
    /// Constructor for the Amazon SQS wrapper.
    /// </summary>
    /// <param name="amazonSQS">The injected Amazon SQS client.</param>
    public SQSWrapper(IAmazonSQS amazonSQS)
    {
        _amazonSQSClient = amazonSQS;
```

```
    }

    /// <summary>
    /// Create a queue with a specific name.
    /// </summary>
    /// <param name="queueName">The name for the queue.</param>
    /// <param name="useFifoQueue">True to use a FIFO queue.</param>
    /// <returns>The url for the queue.</returns>
    public async Task<string> CreateQueueWithName(string queueName, bool
useFifoQueue)
    {
        int maxMessage = 256 * 1024;
        var queueAttributes = new Dictionary<string, string>
        {
            {
                QueueAttributeName.MaximumMessageSize,
                maxMessage.ToString()
            }
        };

        var createQueueRequest = new CreateQueueRequest()
        {
            QueueName = queueName,
            Attributes = queueAttributes
        };

        if (useFifoQueue)
        {
            // Update the name if it is not correct for a FIFO queue.
            if (!queueName.EndsWith(".fifo"))
            {
                createQueueRequest.QueueName = queueName + ".fifo";
            }

            // Add an attribute for a FIFO queue.
            createQueueRequest.Attributes.Add(
                QueueAttributeName.FifoQueue, "true");
        }

        var createResponse = await _amazonSQSClient.CreateQueueAsync(
            new CreateQueueRequest()
            {
                QueueName = queueName
            });
```

```
        return createResponse.QueueUrl;
    }

    /// <summary>
    /// Get the ARN for a queue from its URL.
    /// </summary>
    /// <param name="queueUrl">The URL of the queue.</param>
    /// <returns>The ARN of the queue.</returns>
    public async Task<string> GetQueueArnByUrl(string queueUrl)
    {
        var getAttributesRequest = new GetQueueAttributesRequest()
        {
            QueueUrl = queueUrl,
            AttributeNames = new List<string>() { QueueAttributeName.QueueArn }
        };

        var getAttributesResponse = await _amazonSQSClient.GetQueueAttributesAsync(
            getAttributesRequest);

        return getAttributesResponse.QueueARN;
    }

    /// <summary>
    /// Set the policy attribute of a queue for a topic.
    /// </summary>
    /// <param name="queueArn">The ARN of the queue.</param>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="queueUrl">The url for the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> SetQueuePolicyForTopic(string queueArn, string topicArn,
string queueUrl)
    {
        var queuePolicy = "{" +
                                "\"Version\": \"2012-10-17\"," +
                                "\"Statement\": [{" +
                                     "\"Effect\": \"Allow\"," +
                                     "\"Principal\": {" +
                                         $"\"Service\": " +
                                             "\"sns.amazonaws.com\"" +
                                            "}," +
                                     "\"Action\": \"sqs:SendMessage\"," +
                                     $"\"Resource\": \"{queueArn}\"," +
                                      "\"Condition\": {" +
                                           "\"ArnEquals\": {" +
```

```
                                                $"\"aws:SourceArn\": \"{topicArn}\""
 +
                                            "}" +
                                        "}" +
                                    "}]" +
                                "}";
        var attributesResponse = await _amazonSQSClient.SetQueueAttributesAsync(
            new SetQueueAttributesRequest()
            {
                QueueUrl = queueUrl,
                Attributes = new Dictionary<string, string>() { { "Policy",
 queuePolicy } }
            });
        return attributesResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Receive messages from a queue by its URL.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>The list of messages.</returns>
    public async Task<List<Message>> ReceiveMessagesByUrl(string queueUrl, int
 maxMessages)
    {
        // Setting WaitTimeSeconds to non-zero enables long polling.
        // For information about long polling, see
        // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-short-and-long-polling.html
        var messageResponse = await _amazonSQSClient.ReceiveMessageAsync(
            new ReceiveMessageRequest()
            {
                QueueUrl = queueUrl,
                MaxNumberOfMessages = maxMessages,
                WaitTimeSeconds = 1
            });
        return messageResponse.Messages;
    }

    /// <summary>
    /// Delete a batch of messages from a queue by its url.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>True if successful.</returns>
```

```
    public async Task<bool> DeleteMessageBatchByUrl(string queueUrl, List<Message>
 messages)
    {
        var deleteRequest = new DeleteMessageBatchRequest()
        {
            QueueUrl = queueUrl,
            Entries = new List<DeleteMessageBatchRequestEntry>()
        };
        foreach (var message in messages)
        {
            deleteRequest.Entries.Add(new DeleteMessageBatchRequestEntry()
            {
                ReceiptHandle = message.ReceiptHandle,
                Id = message.MessageId
            });
        }

        var deleteResponse = await
 _amazonSQSClient.DeleteMessageBatchAsync(deleteRequest);

        return deleteResponse.Failed.Any();
    }

    /// <summary>
    /// Delete a queue by its URL.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteQueueByUrl(string queueUrl)
    {
        var deleteResponse = await _amazonSQSClient.DeleteQueueAsync(
            new DeleteQueueRequest()
            {
                QueueUrl = queueUrl
            });
        return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
    }
}
```

Amazon SNS オペレーションをラップするクラスを作成します。

```
/// <summary>
/// Wrapper for Amazon Simple Notification Service (SNS) operations.
/// </summary>
public class SNSWrapper
{
    private readonly IAmazonSimpleNotificationService _amazonSNSClient;

    /// <summary>
    /// Constructor for the Amazon SNS wrapper.
    /// </summary>
    /// <param name="amazonSQS">The injected Amazon SNS client.</param>
    public SNSWrapper(IAmazonSimpleNotificationService amazonSNS)
    {
        _amazonSNSClient = amazonSNS;
    }

    /// <summary>
    /// Create a new topic with a name and specific FIFO and de-duplication
 attributes.
    /// </summary>
    /// <param name="topicName">The name for the topic.</param>
    /// <param name="useFifoTopic">True to use a FIFO topic.</param>
    /// <param name="useContentBasedDeduplication">True to use content-based de-
duplication.</param>
    /// <returns>The ARN of the new topic.</returns>
    public async Task<string> CreateTopicWithName(string topicName, bool
 useFifoTopic, bool useContentBasedDeduplication)
    {
        var createTopicRequest = new CreateTopicRequest()
        {
            Name = topicName,
        };

        if (useFifoTopic)
        {
            // Update the name if it is not correct for a FIFO topic.
            if (!topicName.EndsWith(".fifo"))
            {
                createTopicRequest.Name = topicName + ".fifo";
            }

            // Add the attributes from the method parameters.
            createTopicRequest.Attributes = new Dictionary<string, string>
            {
```

```
                { "FifoTopic", "true" }
            };
            if (useContentBasedDeduplication)
            {
                createTopicRequest.Attributes.Add("ContentBasedDeduplication",
"true");
            }
        }

        var createResponse = await
_amazonSNSClient.CreateTopicAsync(createTopicRequest);
        return createResponse.TopicArn;
    }

    /// <summary>
    /// Subscribe a queue to a topic with optional filters.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="useFifoTopic">The optional filtering policy for the
subscription.</param>
    /// <param name="queueArn">The ARN of the queue.</param>
    /// <returns>The ARN of the new subscription.</returns>
    public async Task<string> SubscribeTopicWithFilter(string topicArn, string?
filterPolicy, string queueArn)
    {
        var subscribeRequest = new SubscribeRequest()
        {
            TopicArn = topicArn,
            Protocol = "sqs",
            Endpoint = queueArn
        };

        if (!string.IsNullOrEmpty(filterPolicy))
        {
            subscribeRequest.Attributes = new Dictionary<string, string>
{ { "FilterPolicy", filterPolicy } };
        }

        var subscribeResponse = await
_amazonSNSClient.SubscribeAsync(subscribeRequest);
        return subscribeResponse.SubscriptionArn;
    }

    /// <summary>
```

```
    /// Publish a message to a topic with an attribute and optional deduplication
 and group IDs.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="message">The message to publish.</param>
    /// <param name="attributeName">The optional attribute for the message.</param>
    /// <param name="attributeValue">The optional attribute value for the message.</
param>
    /// <param name="deduplicationId">The optional deduplication ID for the
 message.</param>
    /// <param name="groupId">The optional group ID for the message.</param>
    /// <returns>The ID of the message published.</returns>
    public async Task<string> PublishToTopicWithAttribute(
        string topicArn,
        string message,
        string? attributeName = null,
        string? attributeValue = null,
        string? deduplicationId = null,
        string? groupId = null)
    {
        var publishRequest = new PublishRequest()
        {
            TopicArn = topicArn,
            Message = message,
            MessageDeduplicationId = deduplicationId,
            MessageGroupId = groupId
        };

        if (attributeValue != null)
        {
            // Add the string attribute if it exists.
            publishRequest.MessageAttributes =
                new Dictionary<string, MessageAttributeValue>
                {
                    { attributeName!, new MessageAttributeValue() { StringValue =
 attributeValue, DataType = "String"} }
                };
        }

        var publishResponse = await _amazonSNSClient.PublishAsync(publishRequest);
        return publishResponse.MessageId;
    }
```

```
    /// <summary>
    /// Unsubscribe from a topic by a subscription ARN.
    /// </summary>
    /// <param name="subscriptionArn">The ARN of the subscription.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> UnsubscribeByArn(string subscriptionArn)
    {
        var unsubscribeResponse = await _amazonSNSClient.UnsubscribeAsync(
            new UnsubscribeRequest()
            {
                SubscriptionArn = subscriptionArn
            });
        return unsubscribeResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete a topic by its topic ARN.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteTopicByArn(string topicArn)
    {
        var deleteResponse = await _amazonSNSClient.DeleteTopicAsync(
            new DeleteTopicRequest()
            {
                TopicArn = topicArn
            });
        return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
    }
}
```

- 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。
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes

- Publish
- ReceiveMessage
- SetQueueAttributes
- Subscribe
- Unsubscribe

サーバーレスサンプル

Amazon SNS トリガーから Lambda 関数を呼び出す

次のコード例は、SNS トピックからメッセージを受信することによってトリガーされるイベントを受け取る Lambda 関数を実装する方法を示しています。この関数はイベントパラメータからメッセージを取得し、各メッセージの内容を記録します。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行する方法を確認してください。

.NET を使用した Lambda での SNS イベントの消費。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SNSEvents;


// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace SnsIntegration;

public class Function
{
    public async Task FunctionHandler(SNSEvent evnt, ILambdaContext context)
```

```
    {
        foreach (var record in evnt.Records)
        {
            await ProcessRecordAsync(record, context);
        }
        context.Logger.LogInformation("done");
    }

    private async Task ProcessRecordAsync(SNSEvent.SNSRecord record, ILambdaContext
 context)
    {
        try
        {
            context.Logger.LogInformation($"Processed record {record.Sns.Message}");

            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
        catch (Exception e)
        {
            //You can use Dead Letter Queue to handle failures. By configuring a
 Lambda DLQ.
            context.Logger.LogError($"An error occurred");
            throw;
        }
    }
}
```

を使用した Amazon SQS の例 AWS SDK for .NET

次のコード例は、Amazon SQS AWS SDK for .NET で 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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。このリンクには GitHub、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が記載されています。

開始方法

Hello Amazon SQS

次のコード例は、Amazon SQS の使用を開始する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSActions;

public static class HelloSQS
{
    static async Task Main(string[] args)
    {
        var sqsClient = new AmazonSQSClient();

        Console.WriteLine($"Hello Amazon SQS! Following are some of your queues:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get the first five queues.
        var response = await sqsClient.ListQueuesAsync(
            new ListQueuesRequest()
            {
                MaxResults = 5
            });

        foreach (var queue in response.QueueUrls)
        {
            Console.WriteLine($"\tQueue Url: {queue}");
            Console.WriteLine();
        }
    }
```

```
}
```

- API の詳細については、「 API リファレンスListQueues」の「」を参照してください。 AWS SDK for .NET

トピック

- アクション
- シナリオ
- サーバーレスサンプル

アクション

`CreateQueue`

次の例は、CreateQueue を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

特定の名前を持つキューを作成します。

```
    /// <summary>
    /// Create a queue with a specific name.
    /// </summary>
    /// <param name="queueName">The name for the queue.</param>
    /// <param name="useFifoQueue">True to use a FIFO queue.</param>
    /// <returns>The url for the queue.</returns>
    public async Task<string> CreateQueueWithName(string queueName, bool
useFifoQueue)
    {
        int maxMessage = 256 * 1024;
        var queueAttributes = new Dictionary<string, string>
        {
```

```
            {
                QueueAttributeName.MaximumMessageSize,
                maxMessage.ToString()
            }
        };

        var createQueueRequest = new CreateQueueRequest()
        {
            QueueName = queueName,
            Attributes = queueAttributes
        };

        if (useFifoQueue)
        {
            // Update the name if it is not correct for a FIFO queue.
            if (!queueName.EndsWith(".fifo"))
            {
                createQueueRequest.QueueName = queueName + ".fifo";
            }

            // Add an attribute for a FIFO queue.
            createQueueRequest.Attributes.Add(
                QueueAttributeName.FifoQueue, "true");
        }

        var createResponse = await _amazonSQSClient.CreateQueueAsync(
            new CreateQueueRequest()
            {
                QueueName = queueName
            });
        return createResponse.QueueUrl;
    }
```

Amazon SQS キューを作成して、そこにメッセージを送信します。

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon;
    using Amazon.SQS;
    using Amazon.SQS.Model;
```

```
    public class CreateSendExample
    {
        // Specify your AWS Region (an example Region is shown).
        private static readonly string QueueName = "Example_Queue";
        private static readonly RegionEndpoint ServiceRegion =
 RegionEndpoint.USWest2;
        private static IAmazonSQS client;

        public static async Task Main()
        {
            client = new AmazonSQSClient(ServiceRegion);
            var createQueueResponse = await CreateQueue(client, QueueName);

            string queueUrl = createQueueResponse.QueueUrl;

            Dictionary<string, MessageAttributeValue> messageAttributes = new
 Dictionary<string, MessageAttributeValue>
            {
                { "Title",   new MessageAttributeValue { DataType = "String",
 StringValue = "The Whistler" } },
                { "Author",  new MessageAttributeValue { DataType = "String",
 StringValue = "John Grisham" } },
                { "WeeksOn", new MessageAttributeValue { DataType = "Number",
 StringValue = "6" } },
            };

            string messageBody = "Information about current NY Times fiction
 bestseller for week of 12/11/2016.";

            var sendMsgResponse = await SendMessage(client, queueUrl, messageBody,
 messageAttributes);
        }

        /// <summary>
        /// Creates a new Amazon SQS queue using the queue name passed to it
        /// in queueName.
        /// </summary>
        /// <param name="client">An SQS client object used to send the message.</
param>
        /// <param name="queueName">A string representing the name of the queue
        /// to create.</param>
        /// <returns>A CreateQueueResponse that contains information about the
        /// newly created queue.</returns>
```

```
        public static async Task<CreateQueueResponse> CreateQueue(IAmazonSQS client,
 string queueName)
        {
            var request = new CreateQueueRequest
            {
                QueueName = queueName,
                Attributes = new Dictionary<string, string>
                {
                    { "DelaySeconds", "60" },
                    { "MessageRetentionPeriod", "86400" },
                },
            };

            var response = await client.CreateQueueAsync(request);
            Console.WriteLine($"Created a queue with URL : {response.QueueUrl}");

            return response;
        }

        /// <summary>
        /// Sends a message to an SQS queue.
        /// </summary>
        /// <param name="client">An SQS client object used to send the message.</
param>
        /// <param name="queueUrl">The URL of the queue to which to send the
        /// message.</param>
        /// <param name="messageBody">A string representing the body of the
        /// message to be sent to the queue.</param>
        /// <param name="messageAttributes">Attributes for the message to be
        /// sent to the queue.</param>
        /// <returns>A SendMessageResponse object that contains information
        /// about the message that was sent.</returns>
        public static async Task<SendMessageResponse> SendMessage(
            IAmazonSQS client,
            string queueUrl,
            string messageBody,
            Dictionary<string, MessageAttributeValue> messageAttributes)
        {
            var sendMessageRequest = new SendMessageRequest
            {
                DelaySeconds = 10,
                MessageAttributes = messageAttributes,
                MessageBody = messageBody,
                QueueUrl = queueUrl,
```

```
            };

            var response = await client.SendMessageAsync(sendMessageRequest);
            Console.WriteLine($"Sent a message with id : {response.MessageId}");

            return response;
        }
    }
```

- API の詳細については、「 API リファレンスCreateQueue」の「」を参照してください。 AWS SDK for .NET

`DeleteMessage`

次の例は、`DeleteMessage` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon SQS キューからメッセージを受信し、メッセージを削除します。

```
        public static async Task Main()
        {
            // If the AWS Region you want to use is different from
            // the AWS Region defined for the default user, supply
            // the specify your AWS Region to the client constructor.
            var client = new AmazonSQSClient();
            string queueName = "Example_Queue";

            var queueUrl = await GetQueueUrl(client, queueName);
            Console.WriteLine($"The SQS queue's URL is {queueUrl}");

            var response = await ReceiveAndDeleteMessage(client, queueUrl);

            Console.WriteLine($"Message: {response.Messages[0]}");
```

```
        }

        /// <summary>
        /// Retrieve the queue URL for the queue named in the queueName
        /// property using the client object.
        /// </summary>
        /// <param name="client">The Amazon SQS client used to retrieve the
        /// queue URL.</param>
        /// <param name="queueName">A string representing  name of the queue
        /// for which to retrieve the URL.</param>
        /// <returns>The URL of the queue.</returns>
        public static async Task<string> GetQueueUrl(IAmazonSQS client, string
queueName)
        {
            var request = new GetQueueUrlRequest
            {
                QueueName = queueName,
            };

            GetQueueUrlResponse response = await client.GetQueueUrlAsync(request);
            return response.QueueUrl;
        }

        /// <summary>
        /// Retrieves the message from the quque at the URL passed in the
        /// queueURL parameters using the client.
        /// </summary>
        /// <param name="client">The SQS client used to retrieve a message.</param>
        /// <param name="queueUrl">The URL of the queue from which to retrieve
        /// a message.</param>
        /// <returns>The response from the call to ReceiveMessageAsync.</returns>
        public static async Task<ReceiveMessageResponse>
ReceiveAndDeleteMessage(IAmazonSQS client, string queueUrl)
        {
            // Receive a single message from the queue.
            var receiveMessageRequest = new ReceiveMessageRequest
            {
                AttributeNames = { "SentTimestamp" },
                MaxNumberOfMessages = 1,
                MessageAttributeNames = { "All" },
                QueueUrl = queueUrl,
                VisibilityTimeout = 0,
                WaitTimeSeconds = 0,
            };
```

```
            var receiveMessageResponse = await
client.ReceiveMessageAsync(receiveMessageRequest);

            // Delete the received message from the queue.
            var deleteMessageRequest = new DeleteMessageRequest
            {
                QueueUrl = queueUrl,
                ReceiptHandle = receiveMessageResponse.Messages[0].ReceiptHandle,
            };

            await client.DeleteMessageAsync(deleteMessageRequest);

            return receiveMessageResponse;
        }
    }
```

- API の詳細については、「 API リファレンスDeleteMessage」の「」を参照してください。 AWS SDK for .NET

`DeleteMessageBatch`

次の例は、DeleteMessageBatch を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete a batch of messages from a queue by its url.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteMessageBatchByUrl(string queueUrl, List<Message>
messages)
```

```
    {
        var deleteRequest = new DeleteMessageBatchRequest()
        {
            QueueUrl = queueUrl,
            Entries = new List<DeleteMessageBatchRequestEntry>()
        };
        foreach (var message in messages)
        {
            deleteRequest.Entries.Add(new DeleteMessageBatchRequestEntry()
            {
                ReceiptHandle = message.ReceiptHandle,
                Id = message.MessageId
            });
        }

        var deleteResponse = await
_amazonSQSClient.DeleteMessageBatchAsync(deleteRequest);

        return deleteResponse.Failed.Any();
    }
```

- API の詳細については、「 API リファレンスDeleteMessageBatch」の「」を参照してください。 AWS SDK for .NET

`DeleteQueue`

次の例は、DeleteQueue を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

URL を使用してキューを削除します。

```
    /// <summary>
    /// Delete a queue by its URL.
    /// </summary>
```

```
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteQueueByUrl(string queueUrl)
    {
        var deleteResponse = await _amazonSQSClient.DeleteQueueAsync(
            new DeleteQueueRequest()
            {
                QueueUrl = queueUrl
            });
        return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスDeleteQueue」の「」を参照してください。 AWS SDK for .NET

`GetQueueAttributes`

次の例は、`GetQueueAttributes` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get the ARN for a queue from its URL.
    /// </summary>
    /// <param name="queueUrl">The URL of the queue.</param>
    /// <returns>The ARN of the queue.</returns>
    public async Task<string> GetQueueArnByUrl(string queueUrl)
    {
        var getAttributesRequest = new GetQueueAttributesRequest()
        {
            QueueUrl = queueUrl,
            AttributeNames = new List<string>() { QueueAttributeName.QueueArn }
        };
```

```
        var getAttributesResponse = await _amazonSQSClient.GetQueueAttributesAsync(
            getAttributesRequest);

        return getAttributesResponse.QueueARN;
    }
```

- API の詳細については、「 API リファレンスGetQueueAttributes」の「」を参照してください。 AWS SDK for .NET

GetQueueUrl

次の例は、GetQueueUrl を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

    public class GetQueueUrl
    {
        /// <summary>
        /// Initializes the Amazon SQS client object and then calls the
        /// GetQueueUrlAsync method to retrieve the URL of an Amazon SQS
        /// queue.
        /// </summary>
        public static async Task Main()
        {
            // If the Amazon SQS message queue is not in the same AWS Region as your
            // default user, you need to provide the AWS Region as a parameter to
the
            // client constructor.
            var client = new AmazonSQSClient();
```

```
            string queueName = "New-Example-Queue";

            try
            {
                var response = await client.GetQueueUrlAsync(queueName);

                if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
                {
                    Console.WriteLine($"The URL for {queueName} is:
{response.QueueUrl}");
                }
            }
            catch (QueueDoesNotExistException ex)
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine($"The queue {queueName} was not found.");
            }
        }
    }
```

- API の詳細については、「 API リファレンスGetQueueUrl」の「」を参照してください。 AWS SDK for .NET

`ReceiveMessage`

次の例は、`ReceiveMessage` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

URL を使用してキューからメッセージを受信します。

```
    /// <summary>
    /// Receive messages from a queue by its URL.
```

```
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>The list of messages.</returns>
    public async Task<List<Message>> ReceiveMessagesByUrl(string queueUrl, int
 maxMessages)
    {
        // Setting WaitTimeSeconds to non-zero enables long polling.
        // For information about long polling, see
        // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-short-and-long-polling.html
        var messageResponse = await _amazonSQSClient.ReceiveMessageAsync(
            new ReceiveMessageRequest()
            {
                QueueUrl = queueUrl,
                MaxNumberOfMessages = maxMessages,
                WaitTimeSeconds = 1
            });
        return messageResponse.Messages;
    }
```

Amazon SQS キューからメッセージを受信し、メッセージを削除します。

```
        public static async Task Main()
        {
            // If the AWS Region you want to use is different from
            // the AWS Region defined for the default user, supply
            // the specify your AWS Region to the client constructor.
            var client = new AmazonSQSClient();
            string queueName = "Example_Queue";

            var queueUrl = await GetQueueUrl(client, queueName);
            Console.WriteLine($"The SQS queue's URL is {queueUrl}");

            var response = await ReceiveAndDeleteMessage(client, queueUrl);

            Console.WriteLine($"Message: {response.Messages[0]}");
        }

        /// <summary>
        /// Retrieve the queue URL for the queue named in the queueName
        /// property using the client object.
        /// </summary>
```

```
        /// <param name="client">The Amazon SQS client used to retrieve the
        /// queue URL.</param>
        /// <param name="queueName">A string representing  name of the queue
        /// for which to retrieve the URL.</param>
        /// <returns>The URL of the queue.</returns>
        public static async Task<string> GetQueueUrl(IAmazonSQS client, string
queueName)
        {
            var request = new GetQueueUrlRequest
            {
                QueueName = queueName,
            };

            GetQueueUrlResponse response = await client.GetQueueUrlAsync(request);
            return response.QueueUrl;
        }

        /// <summary>
        /// Retrieves the message from the quque at the URL passed in the
        /// queueURL parameters using the client.
        /// </summary>
        /// <param name="client">The SQS client used to retrieve a message.</param>
        /// <param name="queueUrl">The URL of the queue from which to retrieve
        /// a message.</param>
        /// <returns>The response from the call to ReceiveMessageAsync.</returns>
        public static async Task<ReceiveMessageResponse>
ReceiveAndDeleteMessage(IAmazonSQS client, string queueUrl)
        {
            // Receive a single message from the queue.
            var receiveMessageRequest = new ReceiveMessageRequest
            {
                AttributeNames = { "SentTimestamp" },
                MaxNumberOfMessages = 1,
                MessageAttributeNames = { "All" },
                QueueUrl = queueUrl,
                VisibilityTimeout = 0,
                WaitTimeSeconds = 0,
            };

            var receiveMessageResponse = await
client.ReceiveMessageAsync(receiveMessageRequest);

            // Delete the received message from the queue.
            var deleteMessageRequest = new DeleteMessageRequest
```

```
            {
                QueueUrl = queueUrl,
                ReceiptHandle = receiveMessageResponse.Messages[0].ReceiptHandle,
            };

            await client.DeleteMessageAsync(deleteMessageRequest);

            return receiveMessageResponse;
        }
    }
```

- API の詳細については、「 API リファレンスReceiveMessage」の「」を参照してください。 AWS SDK for .NET

`SendMessage`

次の例は、SendMessage を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon SQS キューを作成して、そこにメッセージを送信します。

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon;
    using Amazon.SQS;
    using Amazon.SQS.Model;

    public class CreateSendExample
    {
        // Specify your AWS Region (an example Region is shown).
        private static readonly string QueueName = "Example_Queue";
```

```
        private static readonly RegionEndpoint ServiceRegion =
 RegionEndpoint.USWest2;
        private static IAmazonSQS client;

        public static async Task Main()
        {
            client = new AmazonSQSClient(ServiceRegion);
            var createQueueResponse = await CreateQueue(client, QueueName);

            string queueUrl = createQueueResponse.QueueUrl;

            Dictionary<string, MessageAttributeValue> messageAttributes = new
 Dictionary<string, MessageAttributeValue>
            {
                { "Title",   new MessageAttributeValue { DataType = "String",
 StringValue = "The Whistler" } },
                { "Author",  new MessageAttributeValue { DataType = "String",
 StringValue = "John Grisham" } },
                { "WeeksOn", new MessageAttributeValue { DataType = "Number",
 StringValue = "6" } },
            };

            string messageBody = "Information about current NY Times fiction
 bestseller for week of 12/11/2016.";

            var sendMsgResponse = await SendMessage(client, queueUrl, messageBody,
 messageAttributes);
        }

        /// <summary>
        /// Creates a new Amazon SQS queue using the queue name passed to it
        /// in queueName.
        /// </summary>
        /// <param name="client">An SQS client object used to send the message.</
param>
        /// <param name="queueName">A string representing the name of the queue
        /// to create.</param>
        /// <returns>A CreateQueueResponse that contains information about the
        /// newly created queue.</returns>
        public static async Task<CreateQueueResponse> CreateQueue(IAmazonSQS client,
 string queueName)
        {
            var request = new CreateQueueRequest
            {
```

```
                QueueName = queueName,
                Attributes = new Dictionary<string, string>
                {
                    { "DelaySeconds", "60" },
                    { "MessageRetentionPeriod", "86400" },
                },
            };

            var response = await client.CreateQueueAsync(request);
            Console.WriteLine($"Created a queue with URL : {response.QueueUrl}");

            return response;
        }

        /// <summary>
        /// Sends a message to an SQS queue.
        /// </summary>
        /// <param name="client">An SQS client object used to send the message.</
param>
        /// <param name="queueUrl">The URL of the queue to which to send the
        /// message.</param>
        /// <param name="messageBody">A string representing the body of the
        /// message to be sent to the queue.</param>
        /// <param name="messageAttributes">Attributes for the message to be
        /// sent to the queue.</param>
        /// <returns>A SendMessageResponse object that contains information
        /// about the message that was sent.</returns>
        public static async Task<SendMessageResponse> SendMessage(
            IAmazonSQS client,
            string queueUrl,
            string messageBody,
            Dictionary<string, MessageAttributeValue> messageAttributes)
        {
            var sendMessageRequest = new SendMessageRequest
            {
                DelaySeconds = 10,
                MessageAttributes = messageAttributes,
                MessageBody = messageBody,
                QueueUrl = queueUrl,
            };

            var response = await client.SendMessageAsync(sendMessageRequest);
            Console.WriteLine($"Sent a message with id : {response.MessageId}");
```

```
            return response;
        }
    }
```

- API の詳細については、「 API リファレンスSendMessage」の「」を参照してください。 AWS SDK for .NET

`SetQueueAttributes`

次の例は、SetQueueAttributes を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

トピックのキューのポリシー属性を設定します。

```
    /// <summary>
    /// Set the policy attribute of a queue for a topic.
    /// </summary>
    /// <param name="queueArn">The ARN of the queue.</param>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="queueUrl">The url for the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> SetQueuePolicyForTopic(string queueArn, string topicArn,
string queueUrl)
    {
        var queuePolicy = "{" +
                                "\"Version\": \"2012-10-17\"," +
                                "\"Statement\": [{" +
                                     "\"Effect\": \"Allow\"," +
                                     "\"Principal\": {" +
                                         $"\"Service\": " +
                                             "\"sns.amazonaws.com\"" +
                                        "}," +
```

```
                                    "\"Action\": \"sqs:SendMessage\"," +
                                    $"\"Resource\": \"{queueArn}\"," +
                                     "\"Condition\": {" +
                                          "\"ArnEquals\": {" +
                                               $"\"aws:SourceArn\": \"{topicArn}\""
+
                                           "}" +
                                      "}" +
                                "}]" +
                              "}";
        var attributesResponse = await _amazonSQSClient.SetQueueAttributesAsync(
            new SetQueueAttributesRequest()
            {
                QueueUrl = queueUrl,
                Attributes = new Dictionary<string, string>() { { "Policy",
queuePolicy } }
            });
        return attributesResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスSetQueueAttributes」の「」を参照してください。 AWS SDK for .NET

シナリオ

メッセージをキューに発行する

次のコードサンプルは、以下の操作方法を示しています。

- トピック (FIFO または非 FIFO) を作成します。
- フィルターを適用するオプションを使用して、複数のキューをトピックにサブスクライブします。
- メッセージをトピックに発行します。
- キューをポーリングして受信メッセージを確認します。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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のシナリオを実行します。

```
/// <summary>
/// Console application to run a workflow scenario for topics and queues.
/// </summary>
public static class TopicsAndQueues
{
    private static bool _useFifoTopic = false;
    private static bool _useContentBasedDeduplication = false;
    private static string _topicName = null!;
    private static string _topicArn = null!;

    private static readonly int _queueCount = 2;
    private static readonly string[] _queueUrls = new string[_queueCount];
    private static readonly string[] _subscriptionArns = new string[_queueCount];
    private static readonly string[] _tones = { "cheerful", "funny", "serious",
 "sincere" };
    public static SNSWrapper SnsWrapper { get; set; } = null!;
    public static SQSWrapper SqsWrapper { get; set; } = null!;
    public static bool UseConsole { get; set; } = true;
    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon EventBridge.
        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<IAmazonSQS>()
                    .AddAWSService<IAmazonSimpleNotificationService>()
                    .AddTransient<SNSWrapper>()
                    .AddTransient<SQSWrapper>()
            )
```

```
            .Build();

        ServicesSetup(host);
        PrintDescription();

        await RunScenario();

    }

    /// <summary>
    /// Populate the services for use within the console application.
    /// </summary>
    /// <param name="host">The services host.</param>
    private static void ServicesSetup(IHost host)
    {
        SnsWrapper = host.Services.GetRequiredService<SNSWrapper>();
        SqsWrapper = host.Services.GetRequiredService<SQSWrapper>();
    }

    /// <summary>
    /// Run the scenario for working with topics and queues.
    /// </summary>
    /// <returns>True if successful.</returns>
    public static async Task<bool> RunScenario()
    {
        try
        {
            await SetupTopic();

            await SetupQueues();

            await PublishMessages();

            foreach (var queueUrl in _queueUrls)
            {
                var messages = await PollForMessages(queueUrl);
                if (messages.Any())
                {
                    await DeleteMessages(queueUrl, messages);
                }
            }
            await CleanupResources();
```

```
            Console.WriteLine("Messaging with topics and queues workflow is
 complete.");
            return true;
        }
        catch (Exception ex)
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine($"There was a problem running the scenario:
 {ex.Message}");
            await CleanupResources();
            Console.WriteLine(new string('-', 80));
            return false;
        }
    }

    /// <summary>
    /// Print a description for the tasks in the workflow.
    /// </summary>
    /// <returns>Async task.</returns>
    private static void PrintDescription()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Welcome to messaging with topics and queues.");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"In this workflow, you will create an SNS topic and
 subscribe {_queueCount} SQS queues to the topic." +
                          $"\r\nYou can select from several options for configuring
 the topic and the subscriptions for the 2 queues." +
                          $"\r\nYou can then post to the topic and see the results
 in the queues.\r\n");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Set up the SNS topic to be used with the queues.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task<string> SetupTopic()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"SNS topics can be configured as FIFO (First-In-First-
Out)." +
```

```
                        $"\r\nFIFO topics deliver messages in order and support
 deduplication and message filtering." +
                        $"\r\nYou can then post to the topic and see the results
 in the queues.\r\n");

        _useFifoTopic = GetYesNoResponse("Would you like to work with FIFO
 topics?");

        if (_useFifoTopic)
        {
            Console.WriteLine(new string('-', 80));
            _topicName = GetUserResponse("Enter a name for your SNS topic: ",
 "example-topic");
            Console.WriteLine(
                "Because you have selected a FIFO topic, '.fifo' must be appended to
 the topic name.\r\n");

            Console.WriteLine(new string('-', 80));
            Console.WriteLine($"Because you have chosen a FIFO topic, deduplication
 is supported." +
                              $"\r\nDeduplication IDs are either set in the message
 or automatically generated " +
                              $"\r\nfrom content using a hash function.\r\n" +
                              $"\r\nIf a message is successfully published to an SNS
 FIFO topic, any message " +
                              $"\r\npublished and determined to have the same
 deduplication ID, " +
                              $"\r\nwithin the five-minute deduplication interval,
 is accepted but not delivered.\r\n" +
                              $"\r\nFor more information about deduplication, " +
                              $"\r\nsee https://docs.aws.amazon.com/sns/latest/dg/
fifo-message-dedup.html.");

            _useContentBasedDeduplication = GetYesNoResponse("Use content-based
 deduplication instead of entering a deduplication ID?");
            Console.WriteLine(new string('-', 80));
        }

        _topicArn = await SnsWrapper.CreateTopicWithName(_topicName, _useFifoTopic,
 _useContentBasedDeduplication);

        Console.WriteLine($"Your new topic with the name {_topicName}" +
                          $"\r\nand Amazon Resource Name (ARN) {_topicArn}" +
                          $"\r\nhas been created.\r\n");
```

```
        Console.WriteLine(new string('-', 80));
        return _topicArn;
    }

    /// <summary>
    /// Set up the queues.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task SetupQueues()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Now you will create {_queueCount} Amazon Simple Queue
Service (Amazon SQS) queues to subscribe to the topic.");

        // Repeat this section for each queue.
        for (int i = 0; i < _queueCount; i++)
        {
            var queueName = GetUserResponse("Enter a name for an Amazon SQS queue:
", $"example-queue-{i}");
            if (_useFifoTopic)
            {
                // Only explain this once.
                if (i == 0)
                {
                    Console.WriteLine(
                        "Because you have selected a FIFO topic, '.fifo' must be
appended to the queue name.");
                }

                var queueUrl = await SqsWrapper.CreateQueueWithName(queueName,
_useFifoTopic);

                _queueUrls[i] = queueUrl;

                Console.WriteLine($"Your new queue with the name {queueName}" +
                                  $"\r\nand queue URL {queueUrl}" +
                                  $"\r\nhas been created.\r\n");

                if (i == 0)
                {
                    Console.WriteLine(
                        $"The queue URL is used to retrieve the queue ARN,\r\n" +
                        $"which is used to create a subscription.");
```

```
                Console.WriteLine(new string('-', 80));
            }

            var queueArn = await SqsWrapper.GetQueueArnByUrl(queueUrl);

            if (i == 0)
            {
                Console.WriteLine(
                    $"An AWS Identity and Access Management (IAM) policy must be
attached to an SQS queue, enabling it to receive\r\n" +
                    $"messages from an SNS topic");
            }

            await SqsWrapper.SetQueuePolicyForTopic(queueArn, _topicArn,
queueUrl);

            await SetupFilters(i, queueArn, queueName);
        }
    }

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Set up filters with user options for a queue.
/// </summary>
/// <param name="queueCount">The number of this queue.</param>
/// <param name="queueArn">The ARN of the queue.</param>
/// <param name="queueName">The name of the queue.</param>
/// <returns>Async Task.</returns>
public static async Task SetupFilters(int queueCount, string queueArn, string
queueName)
{
    if (_useFifoTopic)
    {
        Console.WriteLine(new string('-', 80));
        // Only explain this once.
        if (queueCount == 0)
        {
            Console.WriteLine(
                "Subscriptions to a FIFO topic can have filters." +
                "If you add a filter to this subscription, then only the
filtered messages " +
                "will be received in the queue.");
```

```
                Console.WriteLine(
                    "For information about message filtering, " +
                    "see https://docs.aws.amazon.com/sns/latest/dg/sns-message-
filtering.html");

                Console.WriteLine(
                    "For this example, you can filter messages by a" +
                    "TONE attribute.");
            }

            var useFilter = GetYesNoResponse($"Filter messages for {queueName}'s
 subscription to the topic?");

            string? filterPolicy = null;
            if (useFilter)
            {
                filterPolicy = CreateFilterPolicy();
            }
            var subscriptionArn = await
 SnsWrapper.SubscribeTopicWithFilter(_topicArn, filterPolicy,
                queueArn);
            _subscriptionArns[queueCount] = subscriptionArn;

            Console.WriteLine(
                $"The queue {queueName} has been subscribed to the topic
 {_topicName} " +
                $"with the subscription ARN {subscriptionArn}");
            Console.WriteLine(new string('-', 80));
        }
    }

    /// <summary>
    /// Use user input to create a filter policy for a subscription.
    /// </summary>
    /// <returns>The serialized filter policy.</returns>
    public static string CreateFilterPolicy()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine(
            $"You can filter messages by one or more of the following" +
            $"TONE attributes.");

        List<string> filterSelections = new List<string>();
```

```
        var selectionNumber = 0;
        do
        {
            Console.WriteLine(
                $"Enter a number to add a TONE filter, or enter 0 to stop adding
 filters.");
            for (int i = 0; i < _tones.Length; i++)
            {
                Console.WriteLine($"\t{i + 1}. {_tones[i]}");
            }

            var selection = GetUserResponse("", filterSelections.Any() ? "0" : "1");
            int.TryParse(selection, out selectionNumber);
            if (selectionNumber > 0 && !
filterSelections.Contains(_tones[selectionNumber - 1]))
            {
                filterSelections.Add(_tones[selectionNumber - 1]);
            }
        } while (selectionNumber != 0);

        var filters = new Dictionary<string, List<string>>
        {
            { "tone", filterSelections }
        };
        string filterPolicy = JsonSerializer.Serialize(filters);
        return filterPolicy;
    }

    /// <summary>
    /// Publish messages using user settings.
    /// </summary>
    /// <returns>Async task.</returns>
    public static async Task PublishMessages()
    {
        Console.WriteLine("Now we can publish messages.");

        var keepSendingMessages = true;
        string? deduplicationId = null;
        string? toneAttribute = null;
        while (keepSendingMessages)
        {
            Console.WriteLine();
```

```
            var message = GetUserResponse("Enter a message to publish.", "This is a
sample message");

            if (_useFifoTopic)
            {
                Console.WriteLine("Because you are using a FIFO topic, you must set
a message group ID." +
                                  "\r\nAll messages within the same group will be
received in the order " +
                                  "they were published.");

                Console.WriteLine();
                var messageGroupId = GetUserResponse("Enter a message group ID for
this message:", "1");

                if (!_useContentBasedDeduplication)
                {
                    Console.WriteLine("Because you are not using content-based
deduplication, " +
                                      "you must enter a deduplication ID.");

                    Console.WriteLine("Enter a deduplication ID for this message.");
                    deduplicationId = GetUserResponse("Enter a deduplication ID for
this message.", "1");
                }

                if (GetYesNoResponse("Add an attribute to this message?"))
                {
                    Console.WriteLine("Enter a number for an attribute.");
                    for (int i = 0; i < _tones.Length; i++)
                    {
                        Console.WriteLine($"\t{i + 1}. {_tones[i]}");
                    }

                    var selection = GetUserResponse("", "1");
                    int.TryParse(selection, out var selectionNumber);

                    if (selectionNumber > 0 && selectionNumber < _tones.Length)
                    {
                        toneAttribute = _tones[selectionNumber - 1];
                    }
                }

                var messageID = await SnsWrapper.PublishToTopicWithAttribute(
```

```
                    _topicArn, message, "tone", toneAttribute, deduplicationId,
messageGroupId);

                Console.WriteLine($"Message published with id {messageID}.");
            }

            keepSendingMessages = GetYesNoResponse("Send another message?", false);
        }
    }

    /// <summary>
    /// Poll for the published messages to see the results of the user's choices.
    /// </summary>
    /// <returns>Async task.</returns>
    public static async Task<List<Message>> PollForMessages(string queueUrl)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Now the SQS queue at {queueUrl} will be polled to
retrieve the messages." +
                          "\r\nPress any key to continue.");
        if (UseConsole)
        {
            Console.ReadLine();
        }

        var moreMessages = true;
        var messages = new List<Message>();
        while (moreMessages)
        {
            var newMessages = await SqsWrapper.ReceiveMessagesByUrl(queueUrl, 10);

            moreMessages = newMessages.Any();
            if (moreMessages)
            {
                messages.AddRange(newMessages);
            }
        }

        Console.WriteLine($"{messages.Count} message(s) were received by the queue
at {queueUrl}.");

        foreach (var message in messages)
        {
            Console.WriteLine("\tMessage:" +
```

```
                                $"\n\t{message.Body}");
        }

        Console.WriteLine(new string('-', 80));
        return messages;
    }

    /// <summary>
    /// Delete the message using handles in a batch.
    /// </summary>
    /// <returns>Async task.</returns>
    public static async Task DeleteMessages(string queueUrl, List<Message> messages)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Now we can delete the messages in this queue in a
batch.");
        await SqsWrapper.DeleteMessageBatchByUrl(queueUrl, messages);
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CleanupResources()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Clean up resources.");

        try
        {
            foreach (var queueUrl in _queueUrls)
            {
                if (!string.IsNullOrEmpty(queueUrl))
                {
                    var deleteQueue =
                        GetYesNoResponse($"Delete queue with url {queueUrl}?");
                    if (deleteQueue)
                    {
                        await SqsWrapper.DeleteQueueByUrl(queueUrl);
                    }
                }
            }
```

```
            foreach (var subscriptionArn in _subscriptionArns)
            {
                if (!string.IsNullOrEmpty(subscriptionArn))
                {
                    await SnsWrapper.UnsubscribeByArn(subscriptionArn);
                }
            }

            var deleteTopic = GetYesNoResponse($"Delete topic {_topicName}?");
            if (deleteTopic)
            {
                await SnsWrapper.DeleteTopicByArn(_topicArn);
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Unable to clean up resources. Here's why:
{ex.Message}.");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Helper method to get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</param>
    /// <param name="defaultAnswer">Optional default answer to use.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question, bool defaultAnswer = true)
    {
        if (UseConsole)
        {
            Console.WriteLine(question);
            var ynResponse = Console.ReadLine();
            var response = ynResponse != null &&
                           ynResponse.Equals("y",
                               StringComparison.InvariantCultureIgnoreCase);
            return response;
        }
        // If not using the console, use the default.
        return defaultAnswer;
    }
```

```
    /// <summary>
    /// Helper method to get a string response from the user through the console.
    /// </summary>
    /// <param name="question">The question string to print on the console.</param>
    /// <param name="defaultAnswer">Optional default answer to use.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static string GetUserResponse(string question, string defaultAnswer)
    {
        if (UseConsole)
        {
            var response = "";
            while (string.IsNullOrEmpty(response))
            {
                Console.WriteLine(question);
                response = Console.ReadLine();
            }
            return response;
        }
        // If not using the console, use the default.
        return defaultAnswer;
    }
}
```

Amazon SQS オペレーションをラップするクラスを作成します。

```
/// <summary>
/// Wrapper for Amazon Simple Queue Service (SQS) operations.
/// </summary>
public class SQSWrapper
{
    private readonly IAmazonSQS _amazonSQSClient;

    /// <summary>
    /// Constructor for the Amazon SQS wrapper.
    /// </summary>
    /// <param name="amazonSQS">The injected Amazon SQS client.</param>
    public SQSWrapper(IAmazonSQS amazonSQS)
    {
        _amazonSQSClient = amazonSQS;
    }
```

```
    /// <summary>
    /// Create a queue with a specific name.
    /// </summary>
    /// <param name="queueName">The name for the queue.</param>
    /// <param name="useFifoQueue">True to use a FIFO queue.</param>
    /// <returns>The url for the queue.</returns>
    public async Task<string> CreateQueueWithName(string queueName, bool
useFifoQueue)
    {
        int maxMessage = 256 * 1024;
        var queueAttributes = new Dictionary<string, string>
        {
            {
                QueueAttributeName.MaximumMessageSize,
                maxMessage.ToString()
            }
        };

        var createQueueRequest = new CreateQueueRequest()
        {
            QueueName = queueName,
            Attributes = queueAttributes
        };

        if (useFifoQueue)
        {
            // Update the name if it is not correct for a FIFO queue.
            if (!queueName.EndsWith(".fifo"))
            {
                createQueueRequest.QueueName = queueName + ".fifo";
            }

            // Add an attribute for a FIFO queue.
            createQueueRequest.Attributes.Add(
                QueueAttributeName.FifoQueue, "true");
        }

        var createResponse = await _amazonSQSClient.CreateQueueAsync(
            new CreateQueueRequest()
            {
                QueueName = queueName
            });
        return createResponse.QueueUrl;
    }
```

```
    /// <summary>
    /// Get the ARN for a queue from its URL.
    /// </summary>
    /// <param name="queueUrl">The URL of the queue.</param>
    /// <returns>The ARN of the queue.</returns>
    public async Task<string> GetQueueArnByUrl(string queueUrl)
    {
        var getAttributesRequest = new GetQueueAttributesRequest()
        {
            QueueUrl = queueUrl,
            AttributeNames = new List<string>() { QueueAttributeName.QueueArn }
        };

        var getAttributesResponse = await _amazonSQSClient.GetQueueAttributesAsync(
            getAttributesRequest);

        return getAttributesResponse.QueueARN;
    }

    /// <summary>
    /// Set the policy attribute of a queue for a topic.
    /// </summary>
    /// <param name="queueArn">The ARN of the queue.</param>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="queueUrl">The url for the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> SetQueuePolicyForTopic(string queueArn, string topicArn,
string queueUrl)
    {
        var queuePolicy = "{" +
                                "\"Version\": \"2012-10-17\"," +
                                "\"Statement\": [{" +
                                     "\"Effect\": \"Allow\"," +
                                     "\"Principal\": {" +
                                         $"\"Service\": " +
                                             "\"sns.amazonaws.com\"" +
                                            "}," +
                                     "\"Action\": \"sqs:SendMessage\"," +
                                     $"\"Resource\": \"{queueArn}\"," +
                                      "\"Condition\": {" +
                                           "\"ArnEquals\": {" +
                                                $"\"aws:SourceArn\": \"{topicArn}\""
+
```

```
                                            "}" +
                                        "}" +
                                    "}]" +
                                "}";
        var attributesResponse = await _amazonSQSClient.SetQueueAttributesAsync(
            new SetQueueAttributesRequest()
            {
                QueueUrl = queueUrl,
                Attributes = new Dictionary<string, string>() { { "Policy",
 queuePolicy } }
            });
        return attributesResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Receive messages from a queue by its URL.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>The list of messages.</returns>
    public async Task<List<Message>> ReceiveMessagesByUrl(string queueUrl, int
 maxMessages)
    {
        // Setting WaitTimeSeconds to non-zero enables long polling.
        // For information about long polling, see
        // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-short-and-long-polling.html
        var messageResponse = await _amazonSQSClient.ReceiveMessageAsync(
            new ReceiveMessageRequest()
            {
                QueueUrl = queueUrl,
                MaxNumberOfMessages = maxMessages,
                WaitTimeSeconds = 1
            });
        return messageResponse.Messages;
    }

    /// <summary>
    /// Delete a batch of messages from a queue by its url.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteMessageBatchByUrl(string queueUrl, List<Message>
 messages)
    {
```

```
        var deleteRequest = new DeleteMessageBatchRequest()
        {
            QueueUrl = queueUrl,
            Entries = new List<DeleteMessageBatchRequestEntry>()
        };
        foreach (var message in messages)
        {
            deleteRequest.Entries.Add(new DeleteMessageBatchRequestEntry()
            {
                ReceiptHandle = message.ReceiptHandle,
                Id = message.MessageId
            });
        }

        var deleteResponse = await
 _amazonSQSClient.DeleteMessageBatchAsync(deleteRequest);

        return deleteResponse.Failed.Any();
    }

    /// <summary>
    /// Delete a queue by its URL.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteQueueByUrl(string queueUrl)
    {
        var deleteResponse = await _amazonSQSClient.DeleteQueueAsync(
            new DeleteQueueRequest()
            {
                QueueUrl = queueUrl
            });
        return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
    }
}
```

Amazon SNS オペレーションをラップするクラスを作成します。

```

/// <summary>
/// Wrapper for Amazon Simple Notification Service (SNS) operations.
/// </summary>
```

```
public class SNSWrapper
{
    private readonly IAmazonSimpleNotificationService _amazonSNSClient;

    /// <summary>
    /// Constructor for the Amazon SNS wrapper.
    /// </summary>
    /// <param name="amazonSQS">The injected Amazon SNS client.</param>
    public SNSWrapper(IAmazonSimpleNotificationService amazonSNS)
    {
        _amazonSNSClient = amazonSNS;
    }

    /// <summary>
    /// Create a new topic with a name and specific FIFO and de-duplication
 attributes.
    /// </summary>
    /// <param name="topicName">The name for the topic.</param>
    /// <param name="useFifoTopic">True to use a FIFO topic.</param>
    /// <param name="useContentBasedDeduplication">True to use content-based de-
duplication.</param>
    /// <returns>The ARN of the new topic.</returns>
    public async Task<string> CreateTopicWithName(string topicName, bool
 useFifoTopic, bool useContentBasedDeduplication)
    {
        var createTopicRequest = new CreateTopicRequest()
        {
            Name = topicName,
        };

        if (useFifoTopic)
        {
            // Update the name if it is not correct for a FIFO topic.
            if (!topicName.EndsWith(".fifo"))
            {
                createTopicRequest.Name = topicName + ".fifo";
            }

            // Add the attributes from the method parameters.
            createTopicRequest.Attributes = new Dictionary<string, string>
            {
                { "FifoTopic", "true" }
            };
            if (useContentBasedDeduplication)
```

```
            {
                createTopicRequest.Attributes.Add("ContentBasedDeduplication",
"true");
            }
        }

        var createResponse = await
_amazonSNSClient.CreateTopicAsync(createTopicRequest);
        return createResponse.TopicArn;
    }

    /// <summary>
    /// Subscribe a queue to a topic with optional filters.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="useFifoTopic">The optional filtering policy for the
subscription.</param>
    /// <param name="queueArn">The ARN of the queue.</param>
    /// <returns>The ARN of the new subscription.</returns>
    public async Task<string> SubscribeTopicWithFilter(string topicArn, string?
filterPolicy, string queueArn)
    {
        var subscribeRequest = new SubscribeRequest()
        {
            TopicArn = topicArn,
            Protocol = "sqs",
            Endpoint = queueArn
        };

        if (!string.IsNullOrEmpty(filterPolicy))
        {
            subscribeRequest.Attributes = new Dictionary<string, string>
{ { "FilterPolicy", filterPolicy } };
        }

        var subscribeResponse = await
_amazonSNSClient.SubscribeAsync(subscribeRequest);
        return subscribeResponse.SubscriptionArn;
    }

    /// <summary>
    /// Publish a message to a topic with an attribute and optional deduplication
and group IDs.
    /// </summary>
```

```
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="message">The message to publish.</param>
    /// <param name="attributeName">The optional attribute for the message.</param>
    /// <param name="attributeValue">The optional attribute value for the message.</
param>
    /// <param name="deduplicationId">The optional deduplication ID for the
 message.</param>
    /// <param name="groupId">The optional group ID for the message.</param>
    /// <returns>The ID of the message published.</returns>
    public async Task<string> PublishToTopicWithAttribute(
        string topicArn,
        string message,
        string? attributeName = null,
        string? attributeValue = null,
        string? deduplicationId = null,
        string? groupId = null)
    {
        var publishRequest = new PublishRequest()
        {
            TopicArn = topicArn,
            Message = message,
            MessageDeduplicationId = deduplicationId,
            MessageGroupId = groupId
        };

        if (attributeValue != null)
        {
            // Add the string attribute if it exists.
            publishRequest.MessageAttributes =
                new Dictionary<string, MessageAttributeValue>
                {
                    { attributeName!, new MessageAttributeValue() { StringValue =
 attributeValue, DataType = "String"} }
                };
        }

        var publishResponse = await _amazonSNSClient.PublishAsync(publishRequest);
        return publishResponse.MessageId;
    }


    /// <summary>
    /// Unsubscribe from a topic by a subscription ARN.
    /// </summary>
```

```
    /// <param name="subscriptionArn">The ARN of the subscription.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> UnsubscribeByArn(string subscriptionArn)
    {
        var unsubscribeResponse = await _amazonSNSClient.UnsubscribeAsync(
            new UnsubscribeRequest()
            {
                SubscriptionArn = subscriptionArn
            });
        return unsubscribeResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete a topic by its topic ARN.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteTopicByArn(string topicArn)
    {
        var deleteResponse = await _amazonSNSClient.DeleteTopicAsync(
            new DeleteTopicRequest()
            {
                TopicArn = topicArn
            });
        return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
    }
}
```

- 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。
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publish
  - ReceiveMessage

- SetQueueAttributes
- Subscribe
- Unsubscribe

サーバーレスサンプル

Amazon SQS トリガーから Lambda 関数を呼び出す

次のコード例では、SQS キューからメッセージを受信することによってトリガーされるイベントを受け取る、Lambda 関数の実装方法を示しています。この関数はイベントパラメータからメッセージを取得し、各メッセージの内容を記録します。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

.NET を使用した Lambda での SQS イベントの消費。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;


// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace SqsIntegrationSampleCode
{
    public async Task FunctionHandler(SQSEvent evnt, ILambdaContext context)
    {
        foreach (var message in evnt.Records)
        {
            await ProcessMessageAsync(message, context);
```

```
        }

        context.Logger.LogInformation("done");
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
 ILambdaContext context)
    {
        try
        {
            context.Logger.LogInformation($"Processed message {message.Body}");

            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
        catch (Exception e)
        {
            //You can use Dead Letter Queue to handle failures. By configuring a
 Lambda DLQ.
            context.Logger.LogError($"An error occurred");
            throw;
        }

    }
}
```

Amazon SQS トリガーを使用した Lambda 関数でのバッチアイテムの失敗のレポート

以下のコード例では、SQS キューからイベントを受け取る Lambda 関数のための、部分的なバッチレスポンスの実装方法を示しています。この関数は、レスポンスとしてバッチアイテムの失敗を報告し、対象のメッセージを後で再試行するよう Lambda に伝えます。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

.NET を使用した Lambda での SQS バッチアイテム失敗のレポート。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali
namespace sqsSample;

public class Function
{
    public async Task<SQSBatchResponse> FunctionHandler(SQSEvent evnt,
 ILambdaContext context)
    {
        List<SQSBatchResponse.BatchItemFailure> batchItemFailures = new
 List<SQSBatchResponse.BatchItemFailure>();
        foreach(var message in evnt.Records)
        {
            try
            {
                //process your message
                await ProcessMessageAsync(message, context);
            }
            catch (System.Exception)
            {
                //Add failed message identifier to the batchItemFailures list
                batchItemFailures.Add(new
 SQSBatchResponse.BatchItemFailure{ItemIdentifier=message.MessageId});
            }
        }
        return new SQSBatchResponse(batchItemFailures);
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
 ILambdaContext context)
    {
        if (String.IsNullOrEmpty(message.Body))
        {
            throw new Exception("No Body in SQS Message.");
        }
```

```
        context.Logger.LogInformation($"Processed message {message.Body}");
        // TODO: Do interesting work based on the new message
        await Task.CompletedTask;
    }
}
```

を使用した Step Functions の例 AWS SDK for .NET

次のコード例は、Step Functions AWS SDK for .NET で 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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。このリンクには GitHub、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が記載されています。

開始方法

Hello Step Functions

次のコード例は、Step Functions の使用を開始する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
namespace StepFunctionsActions;

using Amazon.StepFunctions;
```

```
using Amazon.StepFunctions.Model;

public class HelloStepFunctions
{
    static async Task Main()
    {
        var stepFunctionsClient = new AmazonStepFunctionsClient();

        Console.Clear();
        Console.WriteLine("Welcome to AWS Step Functions");
        Console.WriteLine("Let's list up to 10 of your state machines:");
        var stateMachineListRequest = new ListStateMachinesRequest { MaxResults =
 10 };

        // Get information for up to 10 Step Functions state machines.
        var response = await
 stepFunctionsClient.ListStateMachinesAsync(stateMachineListRequest);

        if (response.StateMachines.Count > 0)
        {
            response.StateMachines.ForEach(stateMachine =>
            {
                Console.WriteLine($"State Machine Name: {stateMachine.Name}\tAmazon
 Resource Name (ARN): {stateMachine.StateMachineArn}");
            });
        }
        else
        {
            Console.WriteLine("\tNo state machines were found.");
        }
    }
}
```

- API の詳細については、「 API リファレンスListStateMachines」の「」を参照してください。AWS SDK for .NET

トピック

- アクション
- シナリオ

アクション

`CreateActivity`

次の例は、CreateActivity を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Create a Step Functions activity using the supplied name.
    /// </summary>
    /// <param name="activityName">The name for the new Step Functions activity.</
param>
    /// <returns>The Amazon Resource Name (ARN) for the new activity.</returns>
    public async Task<string> CreateActivity(string activityName)
    {
        var response = await _amazonStepFunctions.CreateActivityAsync(new
 CreateActivityRequest { Name = activityName });
        return response.ActivityArn;
    }
```

- API の詳細については、「 API リファレンスCreateActivity」の「」を参照してください。AWS SDK for .NET

`CreateStateMachine`

次の例は、CreateStateMachine を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Create a Step Functions state machine.
    /// </summary>
    /// <param name="stateMachineName">Name for the new Step Functions state
    /// machine.</param>
    /// <param name="definition">A JSON string that defines the Step Functions
    /// state machine.</param>
    /// <param name="roleArn">The Amazon Resource Name (ARN) of the role.</param>
    /// <returns></returns>
    public async Task<string> CreateStateMachine(string stateMachineName, string
definition, string roleArn)
    {
        var request = new CreateStateMachineRequest
        {
            Name = stateMachineName,
            Definition = definition,
            RoleArn = roleArn
        };

        var response =
            await _amazonStepFunctions.CreateStateMachineAsync(request);
        return response.StateMachineArn;
    }
```

- API の詳細については、「 API リファレンスCreateStateMachine」の「」を参照してください。 AWS SDK for .NET

`DeleteActivity`

次の例は、`DeleteActivity` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete a Step Machine activity.
    /// </summary>
    /// <param name="activityArn">The Amazon Resource Name (ARN) of
    /// the activity.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteActivity(string activityArn)
    {
        var response = await _amazonStepFunctions.DeleteActivityAsync(new
DeleteActivityRequest { ActivityArn = activityArn });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスDeleteActivity」の「」を参照してください。AWS SDK for .NET

`DeleteStateMachine`

次の例は、`DeleteStateMachine` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete a Step Functions state machine.
```

```
    /// </summary>
    /// <param name="stateMachineArn">The Amazon Resource Name (ARN) of the
    /// state machine.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteStateMachine(string stateMachineArn)
    {
        var response = await _amazonStepFunctions.DeleteStateMachineAsync(new
DeleteStateMachineRequest
        { StateMachineArn = stateMachineArn });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスDeleteStateMachine」の「」を参照してください。 AWS SDK for .NET

`DescribeExecution`

次の例は、`DescribeExecution` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Retrieve information about the specified Step Functions execution.
    /// </summary>
    /// <param name="executionArn">The Amazon Resource Name (ARN) of the
    /// Step Functions execution.</param>
    /// <returns>The API response returned by the API.</returns>
    public async Task<DescribeExecutionResponse> DescribeExecutionAsync(string
executionArn)
    {
        var response = await _amazonStepFunctions.DescribeExecutionAsync(new
DescribeExecutionRequest { ExecutionArn = executionArn });
        return response;
```

```
    }
```

- API の詳細については、「 API リファレンスDescribeExecution」の「」を参照してください。 AWS SDK for .NET

`DescribeStateMachine`

次の例は、DescribeStateMachine を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Retrieve information about the specified Step Functions state machine.
    /// </summary>
    /// <param name="StateMachineArn">The Amazon Resource Name (ARN) of the
    /// Step Functions state machine to retrieve.</param>
    /// <returns>Information about the specified Step Functions state machine.</
returns>
    public async Task<DescribeStateMachineResponse> DescribeStateMachineAsync(string
 StateMachineArn)
    {
        var response = await _amazonStepFunctions.DescribeStateMachineAsync(new
 DescribeStateMachineRequest { StateMachineArn = StateMachineArn });
        return response;
    }
```

- API の詳細については、「 API リファレンスDescribeStateMachine」の「」を参照してください。 AWS SDK for .NET

GetActivityTask

次の例は、GetActivityTask を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
   /// <summary>
   /// Retrieve a task with the specified Step Functions activity
   /// with the specified Amazon Resource Name (ARN).
   /// </summary>
   /// <param name="activityArn">The Amazon Resource Name (ARN) of
   /// the Step Functions activity.</param>
   /// <param name="workerName">The name of the Step Functions worker.</param>
   /// <returns>The response from the Step Functions activity.</returns>
   public async Task<GetActivityTaskResponse> GetActivityTaskAsync(string
activityArn, string workerName)
   {
       var response = await _amazonStepFunctions.GetActivityTaskAsync(new
GetActivityTaskRequest
       { ActivityArn = activityArn, WorkerName = workerName });
       return response;
   }
```

- API の詳細については、「 API リファレンスGetActivityTask」の「」を参照してください。AWS SDK for .NET

ListActivities

次の例は、ListActivities を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
/// List the Step Functions activities for the current account.
/// </summary>
/// <returns>A list of ActivityListItems.</returns>
public async Task<List<ActivityListItem>> ListActivitiesAsync()
{
    var request = new ListActivitiesRequest();
    var activities = new List<ActivityListItem>();

    do
    {
        var response = await _amazonStepFunctions.ListActivitiesAsync(request);

        if (response.NextToken is not null)
        {
            request.NextToken = response.NextToken;
        }

        activities.AddRange(response.Activities);
    }
    while (request.NextToken is not null);

    return activities;
}
```

- API の詳細については、「 API リファレンスListActivities」の「」を参照してください。 AWS SDK for .NET

`ListExecutions`

次の例は、`ListExecutions` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Retrieve information about executions of a Step Functions
    /// state machine.
    /// </summary>
    /// <param name="stateMachineArn">The Amazon Resource Name (ARN) of the
    /// Step Functions state machine.</param>
    /// <returns>A list of ExecutionListItem objects.</returns>
    public async Task<List<ExecutionListItem>> ListExecutionsAsync(string
stateMachineArn)
    {
        var executions = new List<ExecutionListItem>();
        ListExecutionsResponse response;
        var request = new ListExecutionsRequest { StateMachineArn =
stateMachineArn };

        do
        {
            response = await _amazonStepFunctions.ListExecutionsAsync(request);
            executions.AddRange(response.Executions);
            if (response.NextToken is not null)
            {
                request.NextToken = response.NextToken;
            }
        } while (response.NextToken is not null);

        return executions;
    }
```

- API の詳細については、「 API リファレンスListExecutions」の「」を参照してください。AWS SDK for .NET

ListStateMachines

次の例は、ListStateMachines を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Retrieve a list of Step Functions state machines.
    /// </summary>
    /// <returns>A list of StateMachineListItem objects.</returns>
    public async Task<List<StateMachineListItem>> ListStateMachinesAsync()
    {
        var stateMachines = new List<StateMachineListItem>();
        var listStateMachinesPaginator =
            _amazonStepFunctions.Paginators.ListStateMachines(new
ListStateMachinesRequest());

        await foreach (var response in listStateMachinesPaginator.Responses)
        {
            stateMachines.AddRange(response.StateMachines);
        }

        return stateMachines;
    }
```

- API の詳細については、「 API リファレンスListStateMachines」の「」を参照してください。AWS SDK for .NET

SendTaskSuccess

次の例は、SendTaskSuccess を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Indicate that the Step Functions task, indicated by the
    /// task token, has completed successfully.
    /// </summary>
    /// <param name="taskToken">Identifies the task.</param>
    /// <param name="taskResponse">The response received from executing the task.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> SendTaskSuccessAsync(string taskToken, string
 taskResponse)
    {
        var response = await _amazonStepFunctions.SendTaskSuccessAsync(new
 SendTaskSuccessRequest
        { TaskToken = taskToken, Output = taskResponse });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスSendTaskSuccess」の「」を参照してください。AWS SDK for .NET

`StartExecution`

次の例は、`StartExecution` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Start execution of an AWS Step Functions state machine.
    /// </summary>
    /// <param name="executionName">The name to use for the execution.</param>
    /// <param name="executionJson">The JSON string to pass for execution.</param>
    /// <param name="stateMachineArn">The Amazon Resource Name (ARN) of the
    /// Step Functions state machine.</param>
    /// <returns>The Amazon Resource Name (ARN) of the AWS Step Functions
    /// execution.</returns>
    public async Task<string> StartExecutionAsync(string executionJson, string
stateMachineArn)
    {
        var executionRequest = new StartExecutionRequest
        {
            Input = executionJson,
            StateMachineArn = stateMachineArn
        };

        var response = await
_amazonStepFunctions.StartExecutionAsync(executionRequest);
        return response.ExecutionArn;
    }
```

- API の詳細については、「 API リファレンスStartExecution」の「」を参照してください。AWS SDK for .NET

シナリオ

ステートマシンの使用を開始する

次のコードサンプルは、以下の操作方法を示しています。

- アクティビティを作成します。
- 以前に作成したアクティビティをステップとして含む Amazon ステート言語定義からステートマシンを作成します。
- ステートマシンを実行し、ユーザー入力でアクティビティに応答します。
- 実行が完了したら最終ステータスと出力を取得して、リソースをクリーンアップします。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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のシナリオを実行します。

```
global using System.Text.Json;
global using Amazon.StepFunctions;
global using Microsoft.Extensions.Configuration;
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;
global using StepFunctionsActions;
global using LogLevel = Microsoft.Extensions.Logging.LogLevel;



using Amazon.IdentityManagement;
using Amazon.IdentityManagement.Model;
using Amazon.StepFunctions.Model;

namespace StepFunctionsBasics;

public class StepFunctionsBasics
{
    private static ILogger _logger = null!;
    private static IConfigurationRoot _configuration = null!;
    private static IAmazonIdentityManagementService _iamService = null!;
```

```
   static async Task Main(string[] args)
   {
       // Set up dependency injection for AWS Step Functions.
       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<IAmazonStepFunctions>()
                   .AddAWSService<IAmazonIdentityManagementService>()
                   .AddTransient<StepFunctionsWrapper>()
           )
           .Build();

       _logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
           .CreateLogger<StepFunctionsBasics>();

       // Load configuration settings.
       _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 activityName = _configuration["ActivityName"];
       var stateMachineName = _configuration["StateMachineName"];

       var roleName = _configuration["RoleName"];
       var repoBaseDir = _configuration["RepoBaseDir"];
       var jsonFilePath = _configuration["JsonFilePath"];
       var jsonFileName = _configuration["JsonFileName"];

       var uiMethods = new UiMethods();
       var stepFunctionsWrapper =
host.Services.GetRequiredService<StepFunctionsWrapper>();

       _iamService =
host.Services.GetRequiredService<IAmazonIdentityManagementService>();

       // Load definition for the state machine from a JSON file.
```

```
        var stateDefinitionJson = File.ReadAllText($"{repoBaseDir}{jsonFilePath}
{jsonFileName}");

        Console.Clear();
        uiMethods.DisplayOverview();
        uiMethods.PressEnter();

        uiMethods.DisplayTitle("Create activity");
        Console.WriteLine("Let's start by creating an activity.");
        string activityArn;
        string stateMachineArn;

        // Check to see if the activity already exists.
        var activityList = await stepFunctionsWrapper.ListActivitiesAsync();
        var existingActivity = activityList.FirstOrDefault(activity => activity.Name
 == activityName);
        if (existingActivity is not null)
        {
            activityArn = existingActivity.ActivityArn;
            Console.WriteLine($"Activity, {activityName}, already exists.");
        }
        else
        {
            activityArn = await stepFunctionsWrapper.CreateActivity(activityName);
        }

        // Swap the placeholder in the JSON file with the Amazon Resource Name (ARN)
        // of the recently created activity.
        var stateDefinition =
 stateDefinitionJson.Replace("{{DOC_EXAMPLE_ACTIVITY_ARN}}", activityArn);

        uiMethods.DisplayTitle("Create state machine");
        Console.WriteLine("Now we'll create a state machine.");

        // Find or create an IAM role that can be assumed by Step Functions.
        var role = await GetOrCreateStateMachineRole(roleName);

        // See if the state machine already exists.
        var stateMachineList = await stepFunctionsWrapper.ListStateMachinesAsync();
        var existingStateMachine =
            stateMachineList.FirstOrDefault(stateMachine => stateMachine.Name ==
 stateMachineName);
        if (existingStateMachine is not null)
        {
```

```
            Console.WriteLine($"State machine, {stateMachineName}, already
exists.");
            stateMachineArn = existingStateMachine.StateMachineArn;
        }
        else
        {
            // Create the state machine.
            stateMachineArn =
                await stepFunctionsWrapper.CreateStateMachine(stateMachineName,
stateDefinition, role.Arn);
            uiMethods.PressEnter();
        }

        Console.WriteLine("The state machine has been created.");
        var describeStateMachineResponse = await
stepFunctionsWrapper.DescribeStateMachineAsync(stateMachineArn);


Console.WriteLine($"{describeStateMachineResponse.Name}\t{describeStateMachineResponse.Stat
        Console.WriteLine($"Current status: {describeStateMachineResponse.Status}");
        Console.WriteLine($"Amazon Resource Name (ARN) of the role assumed by the
state machine: {describeStateMachineResponse.RoleArn}");

        var userName = string.Empty;
        Console.Write("Before we start the state machine, tell me what should
ChatSFN call you? ");
        userName = Console.ReadLine();

        // Keep asking until the user enters a string value.
        while (string.IsNullOrEmpty(userName))
        {
            Console.Write("Enter your name: ");
            userName = Console.ReadLine();
        }

        var executionJson = @"{""name"": """ + userName + @"""}";

        // Start the state machine execution.
        Console.WriteLine("Now we'll start execution of the state machine.");
        var executionArn = await
stepFunctionsWrapper.StartExecutionAsync(executionJson, stateMachineArn);
        Console.WriteLine("State machine started.");

        Console.WriteLine($"Thank you, {userName}. Now let's get started...");
```

```
        uiMethods.PressEnter();

        uiMethods.DisplayTitle("ChatSFN");

        var isDone = false;
        var response = new GetActivityTaskResponse();
        var taskToken = string.Empty;
        var userChoice = string.Empty;

        while (!isDone)
        {
            response = await stepFunctionsWrapper.GetActivityTaskAsync(activityArn,
"MvpWorker");
            taskToken = response.TaskToken;

            // Parse the returned JSON string.
            var taskJsonResponse = JsonDocument.Parse(response.Input);
            var taskJsonObject = taskJsonResponse.RootElement;
            var message = taskJsonObject.GetProperty("message").GetString();
            var actions =
taskJsonObject.GetProperty("actions").EnumerateArray().Select(x =>
x.ToString()).ToList();
            Console.WriteLine($"\n{message}\n");

            // Prompt the user for another choice.
            Console.WriteLine("ChatSFN: What would you like me to do?");
            actions.ForEach(action => Console.WriteLine($"\t{action}"));
            Console.Write($"\n{userName}, tell me your choice: ");
            userChoice = Console.ReadLine();
            if (userChoice?.ToLower() == "done")
            {
                isDone = true;
            }

            Console.WriteLine($"You have selected: {userChoice}");
            var jsonResponse = @"{""action"": """ + userChoice + @"""}";

            await stepFunctionsWrapper.SendTaskSuccessAsync(taskToken,
jsonResponse);
        }

        await stepFunctionsWrapper.StopExecution(executionArn);
        Console.WriteLine("Now we will wait for the execution to stop.");
        DescribeExecutionResponse executionResponse;
```

```
        do
        {
            executionResponse = await
stepFunctionsWrapper.DescribeExecutionAsync(executionArn);
        } while (executionResponse.Status == ExecutionStatus.RUNNING);

        Console.WriteLine("State machine stopped.");
        uiMethods.PressEnter();

        uiMethods.DisplayTitle("State machine executions");
        Console.WriteLine("Now let's take a look at the execution values for the
state machine.");

        // List the executions.
        var executions = await
stepFunctionsWrapper.ListExecutionsAsync(stateMachineArn);

        uiMethods.DisplayTitle("Step function execution values");
        executions.ForEach(execution =>
        {
            Console.WriteLine($"{execution.Name}\t{execution.StartDate} to
{execution.StopDate}");
        });

        uiMethods.PressEnter();

        // Now delete the state machine and the activity.
        uiMethods.DisplayTitle("Clean up resources");
        Console.WriteLine("Deleting the state machine...");

        await stepFunctionsWrapper.DeleteStateMachine(stateMachineArn);
        Console.WriteLine("State machine deleted.");

        Console.WriteLine("Deleting the activity...");
        await stepFunctionsWrapper.DeleteActivity(activityArn);
        Console.WriteLine("Activity deleted.");

        Console.WriteLine("The Amazon Step Functions scenario is now complete.");
    }

    static async Task<Role> GetOrCreateStateMachineRole(string roleName)
    {
        // Define the policy document for the role.
        var stateMachineRolePolicy = @"{
```

```
         ""Version"": ""2012-10-17"",
        ""Statement"": [{
            ""Sid"": """",
            ""Effect"": ""Allow"",
            ""Principal"": {
                ""Service"": ""states.amazonaws.com""},
            ""Action"": ""sts:AssumeRole""}]}";

        var role = new Role();
        var roleExists = false;

        try
        {
            var getRoleResponse = await _iamService.GetRoleAsync(new GetRoleRequest
 { RoleName = roleName });
            roleExists = true;
            role = getRoleResponse.Role;
        }
        catch (NoSuchEntityException)
        {
            // The role doesn't exist. Create it.
            Console.WriteLine($"Role, {roleName} doesn't exist. Creating it...");
        }

        if (!roleExists)
        {
            var request = new CreateRoleRequest
            {
                RoleName = roleName,
                AssumeRolePolicyDocument = stateMachineRolePolicy,
            };

            var createRoleResponse = await _iamService.CreateRoleAsync(request);
            role = createRoleResponse.Role;
        }

        return role;
    }
}


namespace StepFunctionsBasics;

/// <summary>
```

```
/// Some useful methods to make screen display easier.
/// </summary>
public class UiMethods
{
    private readonly string _sepBar = new('-', Console.WindowWidth);

    /// <summary>
    /// Show information about the scenario.
    /// </summary>
    public void DisplayOverview()
    {
        Console.Clear();
        DisplayTitle("Welcome to the AWS Step Functions Demo");

        Console.WriteLine("This example application will do the following:");
        Console.WriteLine("\t 1. Create an activity.");
        Console.WriteLine("\t 2. Create a state machine.");
        Console.WriteLine("\t 3. Start an execution.");
        Console.WriteLine("\t 4. Run the worker, then stop it.");
        Console.WriteLine("\t 5. List executions.");
        Console.WriteLine("\t 6. Clean up the resources created for the example.");
    }

    /// <summary>
    /// Display a message and wait until the user presses enter.
    /// </summary>
    public void PressEnter()
    {
        Console.Write("\nPress <Enter> to continue.");
        _ = Console.ReadLine();
    }

    /// <summary>
    /// Pad a string with spaces to center it on the console display.
    /// </summary>
    /// <param name="strToCenter"></param>
    /// <returns></returns>
    private 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);
    }
}
```

ステートマシンとアクティビティアクションをラップするクラスを定義します。

```
namespace StepFunctionsActions;

using Amazon.StepFunctions;
using Amazon.StepFunctions.Model;

/// <summary>
/// Wrapper that performs AWS Step Functions actions.
/// </summary>
public class StepFunctionsWrapper
{
    private readonly IAmazonStepFunctions _amazonStepFunctions;

    /// <summary>
    /// The constructor for the StepFunctionsWrapper. Initializes the
    /// client object passed to it.
    /// </summary>
    /// <param name="amazonStepFunctions">An initialized Step Functions client
 object.</param>
    public StepFunctionsWrapper(IAmazonStepFunctions amazonStepFunctions)
    {
        _amazonStepFunctions = amazonStepFunctions;
    }

    /// <summary>
    /// Create a Step Functions activity using the supplied name.
```

```
    /// </summary>
    /// <param name="activityName">The name for the new Step Functions activity.</
param>
    /// <returns>The Amazon Resource Name (ARN) for the new activity.</returns>
    public async Task<string> CreateActivity(string activityName)
    {
        var response = await _amazonStepFunctions.CreateActivityAsync(new
 CreateActivityRequest { Name = activityName });
        return response.ActivityArn;
    }


    /// <summary>
    /// Create a Step Functions state machine.
    /// </summary>
    /// <param name="stateMachineName">Name for the new Step Functions state
    /// machine.</param>
    /// <param name="definition">A JSON string that defines the Step Functions
    /// state machine.</param>
    /// <param name="roleArn">The Amazon Resource Name (ARN) of the role.</param>
    /// <returns></returns>
    public async Task<string> CreateStateMachine(string stateMachineName, string
 definition, string roleArn)
    {
        var request = new CreateStateMachineRequest
        {
            Name = stateMachineName,
            Definition = definition,
            RoleArn = roleArn
        };

        var response =
            await _amazonStepFunctions.CreateStateMachineAsync(request);
        return response.StateMachineArn;
    }


    /// <summary>
    /// Delete a Step Machine activity.
    /// </summary>
    /// <param name="activityArn">The Amazon Resource Name (ARN) of
    /// the activity.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteActivity(string activityArn)
```

```
    {
        var response = await _amazonStepFunctions.DeleteActivityAsync(new
 DeleteActivityRequest { ActivityArn = activityArn });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete a Step Functions state machine.
    /// </summary>
    /// <param name="stateMachineArn">The Amazon Resource Name (ARN) of the
    /// state machine.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteStateMachine(string stateMachineArn)
    {
        var response = await _amazonStepFunctions.DeleteStateMachineAsync(new
 DeleteStateMachineRequest
        { StateMachineArn = stateMachineArn });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Retrieve information about the specified Step Functions execution.
    /// </summary>
    /// <param name="executionArn">The Amazon Resource Name (ARN) of the
    /// Step Functions execution.</param>
    /// <returns>The API response returned by the API.</returns>
    public async Task<DescribeExecutionResponse> DescribeExecutionAsync(string
 executionArn)
    {
        var response = await _amazonStepFunctions.DescribeExecutionAsync(new
 DescribeExecutionRequest { ExecutionArn = executionArn });
        return response;
    }


    /// <summary>
    /// Retrieve information about the specified Step Functions state machine.
    /// </summary>
    /// <param name="StateMachineArn">The Amazon Resource Name (ARN) of the
    /// Step Functions state machine to retrieve.</param>
    /// <returns>Information about the specified Step Functions state machine.</
returns>
```

```
   public async Task<DescribeStateMachineResponse> DescribeStateMachineAsync(string
StateMachineArn)
   {
       var response = await _amazonStepFunctions.DescribeStateMachineAsync(new
DescribeStateMachineRequest { StateMachineArn = StateMachineArn });
       return response;
   }


   /// <summary>
   /// Retrieve a task with the specified Step Functions activity
   /// with the specified Amazon Resource Name (ARN).
   /// </summary>
   /// <param name="activityArn">The Amazon Resource Name (ARN) of
   /// the Step Functions activity.</param>
   /// <param name="workerName">The name of the Step Functions worker.</param>
   /// <returns>The response from the Step Functions activity.</returns>
   public async Task<GetActivityTaskResponse> GetActivityTaskAsync(string
activityArn, string workerName)
   {
       var response = await _amazonStepFunctions.GetActivityTaskAsync(new
GetActivityTaskRequest
       { ActivityArn = activityArn, WorkerName = workerName });
       return response;
   }


   /// <summary>
   /// List the Step Functions activities for the current account.
   /// </summary>
   /// <returns>A list of ActivityListItems.</returns>
   public async Task<List<ActivityListItem>> ListActivitiesAsync()
   {
       var request = new ListActivitiesRequest();
       var activities = new List<ActivityListItem>();

       do
       {
           var response = await _amazonStepFunctions.ListActivitiesAsync(request);

           if (response.NextToken is not null)
           {
               request.NextToken = response.NextToken;
           }
```

```
            activities.AddRange(response.Activities);
        }
        while (request.NextToken is not null);

        return activities;
    }


    /// <summary>
    /// Retrieve information about executions of a Step Functions
    /// state machine.
    /// </summary>
    /// <param name="stateMachineArn">The Amazon Resource Name (ARN) of the
    /// Step Functions state machine.</param>
    /// <returns>A list of ExecutionListItem objects.</returns>
    public async Task<List<ExecutionListItem>> ListExecutionsAsync(string
stateMachineArn)
    {
        var executions = new List<ExecutionListItem>();
        ListExecutionsResponse response;
        var request = new ListExecutionsRequest { StateMachineArn =
stateMachineArn };

        do
        {
            response = await _amazonStepFunctions.ListExecutionsAsync(request);
            executions.AddRange(response.Executions);
            if (response.NextToken is not null)
            {
                request.NextToken = response.NextToken;
            }
        } while (response.NextToken is not null);

        return executions;
    }


    /// <summary>
    /// Retrieve a list of Step Functions state machines.
    /// </summary>
    /// <returns>A list of StateMachineListItem objects.</returns>
    public async Task<List<StateMachineListItem>> ListStateMachinesAsync()
    {
```

```
        var stateMachines = new List<StateMachineListItem>();
        var listStateMachinesPaginator =
            _amazonStepFunctions.Paginators.ListStateMachines(new
 ListStateMachinesRequest());

        await foreach (var response in listStateMachinesPaginator.Responses)
        {
            stateMachines.AddRange(response.StateMachines);
        }

        return stateMachines;
    }


    /// <summary>
    /// Indicate that the Step Functions task, indicated by the
    /// task token, has completed successfully.
    /// </summary>
    /// <param name="taskToken">Identifies the task.</param>
    /// <param name="taskResponse">The response received from executing the task.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> SendTaskSuccessAsync(string taskToken, string
 taskResponse)
    {
        var response = await _amazonStepFunctions.SendTaskSuccessAsync(new
 SendTaskSuccessRequest
        { TaskToken = taskToken, Output = taskResponse });

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Start execution of an AWS Step Functions state machine.
    /// </summary>
    /// <param name="executionName">The name to use for the execution.</param>
    /// <param name="executionJson">The JSON string to pass for execution.</param>
    /// <param name="stateMachineArn">The Amazon Resource Name (ARN) of the
    /// Step Functions state machine.</param>
    /// <returns>The Amazon Resource Name (ARN) of the AWS Step Functions
    /// execution.</returns>
    public async Task<string> StartExecutionAsync(string executionJson, string
 stateMachineArn)
```

```
    {
        var executionRequest = new StartExecutionRequest
        {
            Input = executionJson,
            StateMachineArn = stateMachineArn
        };

        var response = await
 _amazonStepFunctions.StartExecutionAsync(executionRequest);
        return response.ExecutionArn;
    }


    /// <summary>
    /// Stop execution of a Step Functions workflow.
    /// </summary>
    /// <param name="executionArn">The Amazon Resource Name (ARN) of
    /// the Step Functions execution to stop.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> StopExecution(string executionArn)
    {
        var response =
            await _amazonStepFunctions.StopExecutionAsync(new StopExecutionRequest
 { ExecutionArn = executionArn });
        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」の以下のトピックを参照してください。
  - CreateActivity
  - CreateStateMachine
  - DeleteActivity
  - DeleteStateMachine
  - DescribeExecution
  - DescribeStateMachine

- GetActivityTask
- ListActivities
- ListStateMachines
- SendTaskSuccess
- StartExecution
- StopExecution

AWS STS を使用した の例 AWS SDK for .NET

次のコード例は、 AWS SDK for .NET で 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 STS。

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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。このリンクには GitHub、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が記載されています。

トピック
- アクション

アクション

`AssumeRole`

次の例は、`AssumeRole` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

namespace AssumeRoleExample
{
    class AssumeRole
    {
        /// <summary>
        /// This example shows how to use the AWS Security Token
        /// Service (AWS STS) to assume an IAM role.
        ///
        /// NOTE: It is important that the role that will be assumed has a
        /// trust relationship with the account that will assume the role.
        ///
        /// Before you run the example, you need to create the role you want to
        /// assume and have it trust the IAM account that will assume that role.
        ///
        /// See https://docs.aws.amazon.com/IAM/latest/UserGuide/
id_roles_create.html
        /// for help in working with roles.
        /// </summary>

        private static readonly RegionEndpoint REGION = RegionEndpoint.USWest2;

        static async Task Main()
        {
            // Create the SecurityToken client and then display the identity of the
            // default user.
            var roleArnToAssume = "arn:aws:iam::123456789012:role/testAssumeRole";

            var client = new
 Amazon.SecurityToken.AmazonSecurityTokenServiceClient(REGION);

            // Get and display the information about the identity of the default
 user.
            var callerIdRequest = new GetCallerIdentityRequest();
            var caller = await client.GetCallerIdentityAsync(callerIdRequest);
            Console.WriteLine($"Original Caller: {caller.Arn}");

            // Create the request to use with the AssumeRoleAsync call.
```

```
            var assumeRoleReq = new AssumeRoleRequest()
            {
                DurationSeconds = 1600,
                RoleSessionName = "Session1",
                RoleArn = roleArnToAssume
            };

            var assumeRoleRes = await client.AssumeRoleAsync(assumeRoleReq);

            // Now create a new client based on the credentials of the caller
 assuming the role.
            var client2 = new AmazonSecurityTokenServiceClient(credentials:
 assumeRoleRes.Credentials);

            // Get and display information about the caller that has assumed the
 defined role.
            var caller2 = await client2.GetCallerIdentityAsync(callerIdRequest);
            Console.WriteLine($"AssumedRole Caller: {caller2.Arn}");
        }
    }
}
```

- API の詳細については、「 API リファレンスAssumeRole」の「」を参照してください。 AWS SDK for .NET

AWS Support を使用した の例 AWS SDK for .NET

次のコード例は、 AWS SDK for .NET で 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 Support。

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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。このリンクには GitHub、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が記載されています。

開始方法

こんにち AWS Supportは

次のコード例は、 AWS Supportの使用を開始する方法を示しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
using Amazon.AWSSupport;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

public static class HelloSupport
{
    static async Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
 the AWS Support service.
        // Use your AWS profile name, or leave it blank to use the default profile.
        // You must have one of the following AWS Support plans: Business,
 Enterprise On-Ramp, or Enterprise. Otherwise, an exception will be thrown.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonAWSSupport>()
            ).Build();

        // Now the client is available for injection.
        var supportClient = host.Services.GetRequiredService<IAmazonAWSSupport>();

        // You can use await and any of the async methods to get a response.
        var response = await supportClient.DescribeServicesAsync();
        Console.WriteLine($"\tHello AWS Support! There are {response.Services.Count}
 services available.");
    }
}
```

- API の詳細については、「 API リファレンスDescribeServices」の「」を参照してください。 AWS SDK for .NET

トピック

- アクション
- シナリオ

アクション

`AddAttachmentsToSet`

次の例は、AddAttachmentsToSet を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
   /// <summary>
   /// Add an attachment to a set, or create a new attachment set if one does not
exist.
   /// </summary>
   /// <param name="data">The data for the attachment.</param>
   /// <param name="fileName">The file name for the attachment.</param>
   /// <param name="attachmentSetId">Optional setId for the attachment. Creates a
new attachment set if empty.</param>
   /// <returns>The setId of the attachment.</returns>
   public async Task<string> AddAttachmentToSet(MemoryStream data, string fileName,
string? attachmentSetId = null)
   {
       var response = await _amazonSupport.AddAttachmentsToSetAsync(
           new AddAttachmentsToSetRequest
           {
               AttachmentSetId = attachmentSetId,
               Attachments = new List<Attachment>
               {
```

```
                    new Attachment
                    {
                        Data = data,
                        FileName = fileName
                    }
                }
            });
        return response.AttachmentSetId;
    }
```

- API の詳細については、「 API リファレンスAddAttachmentsToSet」の「」を参照してください。 AWS SDK for .NET

`AddCommunicationToCase`

次の例は、AddCommunicationToCase を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Add communication to a case, including optional attachment set ID and CC
email addresses.
    /// </summary>
    /// <param name="caseId">Id for the support case.</param>
    /// <param name="body">Body text of the communication.</param>
    /// <param name="attachmentSetId">Optional Id for an attachment set.</param>
    /// <param name="ccEmailAddresses">Optional list of CC email addresses.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> AddCommunicationToCase(string caseId, string body,
        string? attachmentSetId = null, List<string>? ccEmailAddresses = null)
    {
        var response = await _amazonSupport.AddCommunicationToCaseAsync(
```

```
            new AddCommunicationToCaseRequest()
            {
                CaseId = caseId,
                CommunicationBody = body,
                AttachmentSetId = attachmentSetId,
                CcEmailAddresses = ccEmailAddresses
            });
        return response.Result;
    }
```

- API の詳細については、「 API リファレンスAddCommunicationToCase」の「」を参照してください。 AWS SDK for .NET

`CreateCase`

次の例は、CreateCase を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Create a new support case.
    /// </summary>
    /// <param name="serviceCode">Service code for the new case.</param>
    /// <param name="categoryCode">Category for the new case.</param>
    /// <param name="severityCode">Severity code for the new case.</param>
    /// <param name="subject">Subject of the new case.</param>
    /// <param name="body">Body text of the new case.</param>
    /// <param name="language">Optional language support for your case.
    /// Currently Chinese (“zh”), English ("en"), Japanese ("ja") and Korean (“ko”)
are supported.</param>
    /// <param name="attachmentSetId">Optional Id for an attachment set for the new
case.</param>
```

```
    /// <param name="issueType">Optional issue type for the new case. Options are
"customer-service" or "technical".</param>
    /// <returns>The caseId of the new support case.</returns>
    public async Task<string> CreateCase(string serviceCode, string categoryCode,
string severityCode, string subject,
        string body, string language = "en", string? attachmentSetId = null, string
issueType = "customer-service")
    {
        var response = await _amazonSupport.CreateCaseAsync(
            new CreateCaseRequest()
            {
                ServiceCode = serviceCode,
                CategoryCode = categoryCode,
                SeverityCode = severityCode,
                Subject = subject,
                Language = language,
                AttachmentSetId = attachmentSetId,
                IssueType = issueType,
                CommunicationBody = body
            });
        return response.CaseId;
    }
```

- API の詳細については、「 API リファレンスCreateCase」の「」を参照してください。 AWS SDK for .NET

`DescribeAttachment`

次の例は、`DescribeAttachment` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get description of a specific attachment.
    /// </summary>
    /// <param name="attachmentId">Id of the attachment, usually fetched by
describing the communications of a case.</param>
    /// <returns>The attachment object.</returns>
    public async Task<Attachment> DescribeAttachment(string attachmentId)
    {
        var response = await _amazonSupport.DescribeAttachmentAsync(
            new DescribeAttachmentRequest()
            {
                AttachmentId = attachmentId
            });
        return response.Attachment;
    }
```

- API の詳細については、「 API リファレンスDescribeAttachment」の「」を参照してください。 AWS SDK for .NET

`DescribeCases`

次の例は、`DescribeCases` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get case details for a list of case ids, optionally with date filters.
    /// </summary>
    /// <param name="caseIds">The list of case IDs.</param>
    /// <param name="displayId">Optional display ID.</param>
    /// <param name="includeCommunication">True to include communication. Defaults
to true.</param>
```

```
    /// <param name="includeResolvedCases">True to include resolved cases. Defaults
 to false.</param>
    /// <param name="afterTime">The optional start date for a filtered search.</
param>
    /// <param name="beforeTime">The optional end date for a filtered search.</
param>
    /// <param name="language">Optional language support for your case.
    /// Currently Chinese (“zh”), English ("en"), Japanese ("ja") and Korean (“ko”)
 are supported.</param>
    /// <returns>A list of CaseDetails.</returns>
    public async Task<List<CaseDetails>> DescribeCases(List<string> caseIds, string?
 displayId = null, bool includeCommunication = true,
        bool includeResolvedCases = false, DateTime? afterTime = null, DateTime?
 beforeTime = null,
        string language = "en")
    {
        var results = new List<CaseDetails>();
        var paginateCases = _amazonSupport.Paginators.DescribeCases(
            new DescribeCasesRequest()
            {
                CaseIdList = caseIds,
                DisplayId = displayId,
                IncludeCommunications = includeCommunication,
                IncludeResolvedCases = includeResolvedCases,
                AfterTime = afterTime?.ToString("s"),
                BeforeTime = beforeTime?.ToString("s"),
                Language = language
            });
        // Get the entire list using the paginator.
        await foreach (var cases in paginateCases.Cases)
        {
            results.Add(cases);
        }
        return results;
    }
```

- API の詳細については、「 API リファレンスDescribeCases」の「」を参照してください。 AWS SDK for .NET

DescribeCommunications

次の例は、DescribeCommunications を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Describe the communications for a case, optionally with a date filter.
    /// </summary>
    /// <param name="caseId">The ID of the support case.</param>
    /// <param name="afterTime">The optional start date for a filtered search.</
param>
    /// <param name="beforeTime">The optional end date for a filtered search.</
param>
    /// <returns>The list of communications for the case.</returns>
    public async Task<List<Communication>> DescribeCommunications(string caseId,
 DateTime? afterTime = null, DateTime? beforeTime = null)
    {
        var results = new List<Communication>();
        var paginateCommunications =
 _amazonSupport.Paginators.DescribeCommunications(
            new DescribeCommunicationsRequest()
            {
                CaseId = caseId,
                AfterTime = afterTime?.ToString("s"),
                BeforeTime = beforeTime?.ToString("s")
            });
        // Get the entire list using the paginator.
        await foreach (var communications in paginateCommunications.Communications)
        {
            results.Add(communications);
        }
        return results;
    }
```

- API の詳細については、「 API リファレンスDescribeCommunications」の「」を参照してください。 AWS SDK for .NET

`DescribeServices`

次の例は、`DescribeServices` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get the descriptions of AWS services.
    /// </summary>
    /// <param name="name">Optional language for services.
    /// Currently Chinese (“zh”), English ("en"), Japanese ("ja") and Korean (“ko”)
are supported.</param>
    /// <returns>The list of AWS service descriptions.</returns>
    public async Task<List<Service>> DescribeServices(string language = "en")
    {
        var response = await _amazonSupport.DescribeServicesAsync(
            new DescribeServicesRequest()
            {
                Language = language
            });
        return response.Services;
    }
```

- API の詳細については、「 API リファレンスDescribeServices」の「」を参照してください。 AWS SDK for .NET

DescribeSeverityLevels

次の例は、DescribeSeverityLevels を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
   /// <summary>
   /// Get the descriptions of support severity levels.
   /// </summary>
   /// <param name="name">Optional language for severity levels.
   /// Currently Chinese (“zh”), English ("en"), Japanese ("ja") and Korean (“ko”)
are supported.</param>
   /// <returns>The list of support severity levels.</returns>
   public async Task<List<SeverityLevel>> DescribeSeverityLevels(string language =
"en")
   {
       var response = await _amazonSupport.DescribeSeverityLevelsAsync(
           new DescribeSeverityLevelsRequest()
           {
               Language = language
           });
       return response.SeverityLevels;
   }
```

- API の詳細については、「 API リファレンスDescribeSeverityLevels」の「」を参照してください。 AWS SDK for .NET

ResolveCase

次の例は、ResolveCase を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
/// Resolve a support case by caseId.
/// </summary>
/// <param name="caseId">Id for the support case.</param>
/// <returns>The final status of the case after resolving.</returns>
public async Task<string> ResolveCase(string caseId)
{
    var response = await _amazonSupport.ResolveCaseAsync(
        new ResolveCaseRequest()
        {
            CaseId = caseId
        });
    return response.FinalCaseStatus;
}
```

- API の詳細については、「 API リファレンスResolveCase」の「」を参照してください。AWS SDK for .NET

シナリオ

ケースを開始する

次のコードサンプルは、以下の操作方法を示しています。

- ケースの利用可能なサービスと重要度レベルを取得して表示する方法
- 選択したサービス、カテゴリ、重要度レベルを使用してサポートケースを作成する方法
- 当日のオープンケースのリストを取得して表示する方法
- 新しいケースに添付セットとコミュニケーションを追加する方法
- ケースの新しい添付ファイルとコミュニケーションについて説明する方法

- ケースを解決する方法
- 当日の解決済みケースのリストを取得して表示する方法

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、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のシナリオを実行します。

```
/// <summary>
/// Hello AWS Support example.
/// </summary>
public static class SupportCaseScenario
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.
    To use the AWS Support API, you must have one of the following AWS Support
 plans: Business, Enterprise On-Ramp, or Enterprise.

    This .NET example performs the following tasks:
    1.  Get and display services. Select a service from the list.
    2.  Select a category from the selected service.
    3.  Get and display severity levels and select a severity level from the list.
    4.  Create a support case using the selected service, category, and severity
 level.
    5.  Get and display a list of open support cases for the current day.
    6.  Create an attachment set with a sample text file to add to the case.
    7.  Add a communication with the attachment to the support case.
    8.  List the communications of the support case.
    9.  Describe the attachment set.
    10. Resolve the support case.
    11. Get a list of resolved cases for the current day.
   */

    private static SupportWrapper _supportWrapper = null!;
```

```
   static async Task Main(string[] args)
   {
       // Set up dependency injection for the AWS Support service.
       // 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<IAmazonAWSSupport>(new AWSOptions() { Profile
= "default" })
                   .AddTransient<SupportWrapper>()
           )
           .Build();

       var logger = LoggerFactory.Create(builder =>
       {
           builder.AddConsole();
       }).CreateLogger(typeof(SupportCaseScenario));

       _supportWrapper = host.Services.GetRequiredService<SupportWrapper>();

       Console.WriteLine(new string('-', 80));
       Console.WriteLine("Welcome to the AWS Support case example scenario.");
       Console.WriteLine(new string('-', 80));

       try
       {
           var apiSupported = await _supportWrapper.VerifySubscription();
           if (!apiSupported)
           {
               logger.LogError("You must have a Business, Enterprise On-Ramp, or
Enterprise Support " +
                               "plan to use the AWS Support API. \n\tPlease
upgrade your subscription to run these examples.");
               return;
           }

           var service = await DisplayAndSelectServices();

           var category = DisplayAndSelectCategories(service);
```

```
            var severityLevel = await DisplayAndSelectSeverity();

            var caseId = await CreateSupportCase(service, category, severityLevel);

            await DescribeTodayOpenCases();

            var attachmentSetId = await CreateAttachmentSet();

            await AddCommunicationToCase(attachmentSetId, caseId);

            var attachmentId = await ListCommunicationsForCase(caseId);

            await DescribeCaseAttachment(attachmentId);

            await ResolveCase(caseId);

            await DescribeTodayResolvedCases();

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("AWS Support case example scenario complete.");
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            logger.LogError(ex, "There was a problem executing the scenario.");
        }
    }

    /// <summary>
    /// List some available services from AWS Support, and select a service for the
example.
    /// </summary>
    /// <returns>The selected service.</returns>
    private static async Task<Service> DisplayAndSelectServices()
    {
        Console.WriteLine(new string('-', 80));
        var services = await _supportWrapper.DescribeServices();
        Console.WriteLine($"AWS Support client returned {services.Count}
services.");

        Console.WriteLine($"1. Displaying first 10 services:");
        for (int i = 0; i < 10 && i < services.Count; i++)
        {
```

```
            Console.WriteLine($"\t{i + 1}. {services[i].Name}");
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > services.Count)
        {
            Console.WriteLine(
                "Select an example support service by entering a number from the
preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }
        Console.WriteLine(new string('-', 80));

        return services[choiceNumber - 1];
    }

    /// <summary>
    /// List the available categories for a service and select a category for the
example.
    /// </summary>
    /// <param name="service">Service to use for displaying categories.</param>
    /// <returns>The selected category.</returns>
    private static Category DisplayAndSelectCategories(Service service)
    {
        Console.WriteLine(new string('-', 80));

        Console.WriteLine($"2. Available support categories for Service
\"{service.Name}\":");
        for (int i = 0; i < service.Categories.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {service.Categories[i].Name}");
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > service.Categories.Count)
        {
            Console.WriteLine(
                "Select an example support category by entering a number from the
preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }
```

```
        Console.WriteLine(new string('-', 80));

        return service.Categories[choiceNumber - 1];
    }

    /// <summary>
    /// List available severity levels from AWS Support, and select a level for the
example.
    /// </summary>
    /// <returns>The selected severity level.</returns>
    private static async Task<SeverityLevel> DisplayAndSelectSeverity()
    {
        Console.WriteLine(new string('-', 80));
        var severityLevels = await _supportWrapper.DescribeSeverityLevels();

        Console.WriteLine($"3. Get and display available severity levels:");
        for (int i = 0; i < 10 && i < severityLevels.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {severityLevels[i].Name}");
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > severityLevels.Count)
        {
            Console.WriteLine(
                "Select an example severity level by entering a number from the
preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }
        Console.WriteLine(new string('-', 80));

        return severityLevels[choiceNumber - 1];
    }

    /// <summary>
    /// Create an example support case.
    /// </summary>
    /// <param name="service">Service to use for the new case.</param>
    /// <param name="category">Category to use for the new case.</param>
    /// <param name="severity">Severity to use for the new case.</param>
    /// <returns>The caseId of the new support case.</returns>
    private static async Task<string> CreateSupportCase(Service service,
        Category category, SeverityLevel severity)
```

```
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"4. Create an example support case" +
                          $" with the following settings:" +
                          $" \n\tService: {service.Name}, Category: {category.Name}
" +
                          $"and Severity Level: {severity.Name}.");
        var caseId = await _supportWrapper.CreateCase(service.Code, category.Code,
severity.Code,
            "Example case for testing, ignore.", "This is my example support
case.");

        Console.WriteLine($"\tNew case created with ID {caseId}");

        Console.WriteLine(new string('-', 80));

        return caseId;
    }

    /// <summary>
    /// List open cases for the current day.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task DescribeTodayOpenCases()
    {
        Console.WriteLine($"5. List the open support cases for the current day.");
        // Describe the cases. If it is empty, try again and allow time for the new
case to appear.
        List<CaseDetails> currentOpenCases = null!;
        while (currentOpenCases == null || currentOpenCases.Count == 0)
        {
            Thread.Sleep(1000);
            currentOpenCases = await _supportWrapper.DescribeCases(
                new List<string>(),
                null,
                false,
                false,
                DateTime.UtcNow.Date,
                DateTime.UtcNow);
        }

        foreach (var openCase in currentOpenCases)
        {
```

```
            Console.WriteLine($"\tCase: {openCase.CaseId} created
 {openCase.TimeCreated}");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Create an attachment set for a support case.
    /// </summary>
    /// <returns>The attachment set id.</returns>
    private static async Task<string> CreateAttachmentSet()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"6. Create an attachment set for a support case.");
        var fileName = "example_attachment.txt";

        // Create the file if it does not already exist.
        if (!File.Exists(fileName))
        {
            await using StreamWriter sw = File.CreateText(fileName);
            await sw.WriteLineAsync(
                "This is a sample file for attachment to a support case.");
        }

        await using var ms = new MemoryStream(await
 File.ReadAllBytesAsync(fileName));

        var attachmentSetId = await _supportWrapper.AddAttachmentToSet(
            ms,
            fileName);

        Console.WriteLine($"\tNew attachment set created with id: \n
\t{attachmentSetId.Substring(0, 65)}...");

        Console.WriteLine(new string('-', 80));

        return attachmentSetId;
    }

    /// <summary>
    /// Add an attachment set and communication to a case.
    /// </summary>
    /// <param name="attachmentSetId">Id of the attachment set.</param>
```

```
    /// <param name="caseId">Id of the case to receive the attachment set.</param>
    /// <returns>Async task.</returns>
    private static async Task AddCommunicationToCase(string attachmentSetId, string
caseId)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"7. Add attachment set and communication to {caseId}.");

        await _supportWrapper.AddCommunicationToCase(
            caseId,
            "This is an example communication added to a support case.",
            attachmentSetId);

        Console.WriteLine($"\tNew attachment set and communication added to
{caseId}");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List the communications for a case.
    /// </summary>
    /// <param name="caseId">Id of the case to describe.</param>
    /// <returns>An attachment id.</returns>
    private static async Task<string> ListCommunicationsForCase(string caseId)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"8. List communications for case {caseId}.");

        var communications = await _supportWrapper.DescribeCommunications(caseId);
        var attachmentId = "";
        foreach (var communication in communications)
        {
            Console.WriteLine(
                $"\tCommunication created on: {communication.TimeCreated} has
{communication.AttachmentSet.Count} attachments.");
            if (communication.AttachmentSet.Any())
            {
                attachmentId = communication.AttachmentSet.First().AttachmentId;
            }
        }

        Console.WriteLine(new string('-', 80));
        return attachmentId;
```

```
    }

    /// <summary>
    /// Describe an attachment by id.
    /// </summary>
    /// <param name="attachmentId">Id of the attachment to describe.</param>
    /// <returns>Async task.</returns>
    private static async Task DescribeCaseAttachment(string attachmentId)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"9. Describe the attachment set.");

        var attachment = await _supportWrapper.DescribeAttachment(attachmentId);
        var data = Encoding.ASCII.GetString(attachment.Data.ToArray());
        Console.WriteLine($"\tAttachment includes {attachment.FileName} with data:
\n\t{data}");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Resolve the support case.
    /// </summary>
    /// <param name="caseId">Id of the case to resolve.</param>
    /// <returns>Async task.</returns>
    private static async Task ResolveCase(string caseId)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"10. Resolve case {caseId}.");

        var status = await _supportWrapper.ResolveCase(caseId);
        Console.WriteLine($"\tCase {caseId} has final status {status}");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List resolved cases for the current day.
    /// </summary>
    /// <returns>Async Task.</returns>
    private static async Task DescribeTodayResolvedCases()
    {
        Console.WriteLine(new string('-', 80));
```

```
        Console.WriteLine($"11. List the resolved support cases for the current
 day.");
        var currentCases = await _supportWrapper.DescribeCases(
            new List<string>(),
            null,
            false,
            true,
            DateTime.UtcNow.Date,
            DateTime.UtcNow);

        foreach (var currentCase in currentCases)
        {
            if (currentCase.Status == "resolved")
            {
                Console.WriteLine(
                    $"\tCase: {currentCase.CaseId}: status {currentCase.Status}");
            }
        }

        Console.WriteLine(new string('-', 80));
    }
}
```

AWS Support アクションのシナリオで使用されるラッパーメソッド。

```
/// <summary>
/// Wrapper methods to use AWS Support for working with support cases.
/// </summary>
public class SupportWrapper
{
    private readonly IAmazonAWSSupport _amazonSupport;
    public SupportWrapper(IAmazonAWSSupport amazonSupport)
    {
        _amazonSupport = amazonSupport;
    }


    /// <summary>
    /// Get the descriptions of AWS services.
    /// </summary>
    /// <param name="name">Optional language for services.
```

```
    /// Currently Chinese (“zh”), English ("en"), Japanese ("ja") and Korean (“ko”)
are supported.</param>
    /// <returns>The list of AWS service descriptions.</returns>
    public async Task<List<Service>> DescribeServices(string language = "en")
    {
        var response = await _amazonSupport.DescribeServicesAsync(
            new DescribeServicesRequest()
            {
                Language = language
            });
        return response.Services;
    }



    /// <summary>
    /// Get the descriptions of support severity levels.
    /// </summary>
    /// <param name="name">Optional language for severity levels.
    /// Currently Chinese (“zh”), English ("en"), Japanese ("ja") and Korean (“ko”)
are supported.</param>
    /// <returns>The list of support severity levels.</returns>
    public async Task<List<SeverityLevel>> DescribeSeverityLevels(string language =
"en")
    {
        var response = await _amazonSupport.DescribeSeverityLevelsAsync(
            new DescribeSeverityLevelsRequest()
            {
                Language = language
            });
        return response.SeverityLevels;
    }



    /// <summary>
    /// Create a new support case.
    /// </summary>
    /// <param name="serviceCode">Service code for the new case.</param>
    /// <param name="categoryCode">Category for the new case.</param>
    /// <param name="severityCode">Severity code for the new case.</param>
    /// <param name="subject">Subject of the new case.</param>
    /// <param name="body">Body text of the new case.</param>
    /// <param name="language">Optional language support for your case.
```

```
   /// Currently Chinese (“zh”), English ("en"), Japanese ("ja") and Korean (“ko”)
are supported.</param>
   /// <param name="attachmentSetId">Optional Id for an attachment set for the new
case.</param>
   /// <param name="issueType">Optional issue type for the new case. Options are
"customer-service" or "technical".</param>
   /// <returns>The caseId of the new support case.</returns>
   public async Task<string> CreateCase(string serviceCode, string categoryCode,
string severityCode, string subject,
       string body, string language = "en", string? attachmentSetId = null, string
issueType = "customer-service")
   {
       var response = await _amazonSupport.CreateCaseAsync(
           new CreateCaseRequest()
           {
               ServiceCode = serviceCode,
               CategoryCode = categoryCode,
               SeverityCode = severityCode,
               Subject = subject,
               Language = language,
               AttachmentSetId = attachmentSetId,
               IssueType = issueType,
               CommunicationBody = body
           });
       return response.CaseId;
   }



   /// <summary>
   /// Add an attachment to a set, or create a new attachment set if one does not
exist.
   /// </summary>
   /// <param name="data">The data for the attachment.</param>
   /// <param name="fileName">The file name for the attachment.</param>
   /// <param name="attachmentSetId">Optional setId for the attachment. Creates a
new attachment set if empty.</param>
   /// <returns>The setId of the attachment.</returns>
   public async Task<string> AddAttachmentToSet(MemoryStream data, string fileName,
string? attachmentSetId = null)
   {
       var response = await _amazonSupport.AddAttachmentsToSetAsync(
           new AddAttachmentsToSetRequest
           {
```

```
                AttachmentSetId = attachmentSetId,
                Attachments = new List<Attachment>
                {
                    new Attachment
                    {
                        Data = data,
                        FileName = fileName
                    }
                }
            });
        return response.AttachmentSetId;
    }



    /// <summary>
    /// Get description of a specific attachment.
    /// </summary>
    /// <param name="attachmentId">Id of the attachment, usually fetched by
describing the communications of a case.</param>
    /// <returns>The attachment object.</returns>
    public async Task<Attachment> DescribeAttachment(string attachmentId)
    {
        var response = await _amazonSupport.DescribeAttachmentAsync(
            new DescribeAttachmentRequest()
            {
                AttachmentId = attachmentId
            });
        return response.Attachment;
    }



    /// <summary>
    /// Add communication to a case, including optional attachment set ID and CC
email addresses.
    /// </summary>
    /// <param name="caseId">Id for the support case.</param>
    /// <param name="body">Body text of the communication.</param>
    /// <param name="attachmentSetId">Optional Id for an attachment set.</param>
    /// <param name="ccEmailAddresses">Optional list of CC email addresses.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> AddCommunicationToCase(string caseId, string body,
        string? attachmentSetId = null, List<string>? ccEmailAddresses = null)
```

```
    {
        var response = await _amazonSupport.AddCommunicationToCaseAsync(
            new AddCommunicationToCaseRequest()
            {
                CaseId = caseId,
                CommunicationBody = body,
                AttachmentSetId = attachmentSetId,
                CcEmailAddresses = ccEmailAddresses
            });
        return response.Result;
    }



    /// <summary>
    /// Describe the communications for a case, optionally with a date filter.
    /// </summary>
    /// <param name="caseId">The ID of the support case.</param>
    /// <param name="afterTime">The optional start date for a filtered search.</
param>
    /// <param name="beforeTime">The optional end date for a filtered search.</
param>
    /// <returns>The list of communications for the case.</returns>
    public async Task<List<Communication>> DescribeCommunications(string caseId,
 DateTime? afterTime = null, DateTime? beforeTime = null)
    {
        var results = new List<Communication>();
        var paginateCommunications =
 _amazonSupport.Paginators.DescribeCommunications(
            new DescribeCommunicationsRequest()
            {
                CaseId = caseId,
                AfterTime = afterTime?.ToString("s"),
                BeforeTime = beforeTime?.ToString("s")
            });
        // Get the entire list using the paginator.
        await foreach (var communications in paginateCommunications.Communications)
        {
            results.Add(communications);
        }
        return results;
    }
```

```
    /// <summary>
    /// Get case details for a list of case ids, optionally with date filters.
    /// </summary>
    /// <param name="caseIds">The list of case IDs.</param>
    /// <param name="displayId">Optional display ID.</param>
    /// <param name="includeCommunication">True to include communication. Defaults
 to true.</param>
    /// <param name="includeResolvedCases">True to include resolved cases. Defaults
 to false.</param>
    /// <param name="afterTime">The optional start date for a filtered search.</
param>
    /// <param name="beforeTime">The optional end date for a filtered search.</
param>
    /// <param name="language">Optional language support for your case.
    /// Currently Chinese (“zh”), English ("en"), Japanese ("ja") and Korean (“ko”)
 are supported.</param>
    /// <returns>A list of CaseDetails.</returns>
    public async Task<List<CaseDetails>> DescribeCases(List<string> caseIds, string?
 displayId = null, bool includeCommunication = true,
        bool includeResolvedCases = false, DateTime? afterTime = null, DateTime?
 beforeTime = null,
        string language = "en")
    {
        var results = new List<CaseDetails>();
        var paginateCases = _amazonSupport.Paginators.DescribeCases(
            new DescribeCasesRequest()
            {
                CaseIdList = caseIds,
                DisplayId = displayId,
                IncludeCommunications = includeCommunication,
                IncludeResolvedCases = includeResolvedCases,
                AfterTime = afterTime?.ToString("s"),
                BeforeTime = beforeTime?.ToString("s"),
                Language = language
            });
        // Get the entire list using the paginator.
        await foreach (var cases in paginateCases.Cases)
        {
            results.Add(cases);
        }
        return results;
    }
```

```
    /// <summary>
    /// Resolve a support case by caseId.
    /// </summary>
    /// <param name="caseId">Id for the support case.</param>
    /// <returns>The final status of the case after resolving.</returns>
    public async Task<string> ResolveCase(string caseId)
    {
        var response = await _amazonSupport.ResolveCaseAsync(
            new ResolveCaseRequest()
            {
                CaseId = caseId
            });
        return response.FinalCaseStatus;
    }


    /// <summary>
    /// Verify the support level for AWS Support API access.
    /// </summary>
    /// <returns>True if the subscription level supports API access.</returns>
    public async Task<bool> VerifySubscription()
    {
        try
        {
            var response = await _amazonSupport.DescribeServicesAsync(
                new DescribeServicesRequest()
                {
                    Language = "en"
                });
            return response.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (Amazon.AWSSupport.AmazonAWSSupportException ex)
        {
            if (ex.ErrorCode == "SubscriptionRequiredException")
            {
                return false;
            }
            else 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。
  - AddAttachmentsToSet
  - AddCommunicationToCase
  - CreateCase
  - DescribeAttachment
  - DescribeCases
  - DescribeCommunications
  - DescribeServices
  - DescribeSeverityLevels
  - ResolveCase

を使用した Amazon Transcribe の例 AWS SDK for .NET

次のコード例は、Amazon Transcribe AWS SDK for .NET で 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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。このリンクには GitHub、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が記載されています。

トピック

- アクション

アクション

`CreateVocabulary`

次の例は、`CreateVocabulary` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Create a custom vocabulary using a list of phrases. Custom vocabularies
    /// improve transcription accuracy for one or more specific words.
    /// </summary>
    /// <param name="languageCode">The language code of the vocabulary.</param>
    /// <param name="phrases">Phrases to use in the vocabulary.</param>
    /// <param name="vocabularyName">Name for the vocabulary.</param>
    /// <returns>The state of the custom vocabulary.</returns>
    public async Task<VocabularyState> CreateCustomVocabulary(LanguageCode
languageCode,
        List<string> phrases, string vocabularyName)
    {
        var response = await _amazonTranscribeService.CreateVocabularyAsync(
            new CreateVocabularyRequest
            {
                LanguageCode = languageCode,
                Phrases = phrases,
                VocabularyName = vocabularyName
            });
        return response.VocabularyState;
    }
```

- API の詳細については、「 API リファレンスCreateVocabulary」の「」を参照してください。AWS SDK for .NET

`DeleteMedicalTranscriptionJob`

次の例は、DeleteMedicalTranscriptionJob を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete a medical transcription job. Also deletes the transcript associated
 with the job.
    /// </summary>
    /// <param name="jobName">Name of the medical transcription job to delete.</
param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteMedicalTranscriptionJob(string jobName)
    {
        var response = await
 _amazonTranscribeService.DeleteMedicalTranscriptionJobAsync(
            new DeleteMedicalTranscriptionJobRequest()
            {
                MedicalTranscriptionJobName = jobName
            });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスDeleteMedicalTranscriptionJob」の「」を参照してください。 AWS SDK for .NET

`DeleteTranscriptionJob`

次の例は、DeleteTranscriptionJob を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete a transcription job. Also deletes the transcript associated with the
job.
    /// </summary>
    /// <param name="jobName">Name of the transcription job to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteTranscriptionJob(string jobName)
    {
        var response = await _amazonTranscribeService.DeleteTranscriptionJobAsync(
            new DeleteTranscriptionJobRequest()
            {
                TranscriptionJobName = jobName
            });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスDeleteTranscriptionJob」の「」を参照してください。 AWS SDK for .NET

`DeleteVocabulary`

次の例は、`DeleteVocabulary` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete an existing custom vocabulary.
    /// </summary>
    /// <param name="vocabularyName">Name of the vocabulary to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteCustomVocabulary(string vocabularyName)
    {
        var response = await _amazonTranscribeService.DeleteVocabularyAsync(
            new DeleteVocabularyRequest
            {
                VocabularyName = vocabularyName
            });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API の詳細については、「 API リファレンスDeleteVocabulary」の「」を参照してください。AWS SDK for .NET

`GetTranscriptionJob`

次の例は、GetTranscriptionJob を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get details about a transcription job.
    /// </summary>
    /// <param name="jobName">A unique name for the transcription job.</param>
    /// <returns>A TranscriptionJob instance with information on the requested
job.</returns>
    public async Task<TranscriptionJob> GetTranscriptionJob(string jobName)
```

```
    {
        var response = await _amazonTranscribeService.GetTranscriptionJobAsync(
            new GetTranscriptionJobRequest()
            {
                TranscriptionJobName = jobName
            });
        return response.TranscriptionJob;
    }
```

- API の詳細については、「 API リファレンスGetTranscriptionJob」の「」を参照してください。 AWS SDK for .NET

`GetVocabulary`

次の例は、`GetVocabulary` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get information about a custom vocabulary.
    /// </summary>
    /// <param name="vocabularyName">Name of the vocabulary.</param>
    /// <returns>The state of the custom vocabulary.</returns>
    public async Task<VocabularyState> GetCustomVocabulary(string vocabularyName)
    {
        var response = await _amazonTranscribeService.GetVocabularyAsync(
            new GetVocabularyRequest()
            {
                VocabularyName = vocabularyName
            });
        return response.VocabularyState;
    }
```

- API の詳細については、「 API リファレンスGetVocabulary」の「」を参照してください。 AWS SDK for .NET

`ListMedicalTranscriptionJobs`

次の例は、`ListMedicalTranscriptionJobs` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
   /// <summary>
   /// List medical transcription jobs, optionally with a name filter.
   /// </summary>
   /// <param name="jobNameContains">Optional name filter for the medical
transcription jobs.</param>
   /// <returns>A list of summaries about medical transcription jobs.</returns>
   public async Task<List<MedicalTranscriptionJobSummary>>
ListMedicalTranscriptionJobs(
       string? jobNameContains = null)
   {
       var response = await
_amazonTranscribeService.ListMedicalTranscriptionJobsAsync(
           new ListMedicalTranscriptionJobsRequest()
           {
               JobNameContains = jobNameContains
           });
       return response.MedicalTranscriptionJobSummaries;
   }
```

- API の詳細については、「 API リファレンスListMedicalTranscriptionJobs」の「」を参照してください。 AWS SDK for .NET

`ListTranscriptionJobs`

次の例は、`ListTranscriptionJobs` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// List transcription jobs, optionally with a name filter.
    /// </summary>
    /// <param name="jobNameContains">Optional name filter for the transcription
jobs.</param>
    /// <returns>A list of transcription job summaries.</returns>
    public async Task<List<TranscriptionJobSummary>> ListTranscriptionJobs(string?
jobNameContains = null)
    {
        var response = await _amazonTranscribeService.ListTranscriptionJobsAsync(
            new ListTranscriptionJobsRequest()
            {
                JobNameContains = jobNameContains
            });
        return response.TranscriptionJobSummaries;
    }
```

- API の詳細については、「 API リファレンスListTranscriptionJobs」の「」を参照してください。 AWS SDK for .NET

`ListVocabularies`

次の例は、`ListVocabularies` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
   /// <summary>
   /// List custom vocabularies for the current account. Optionally specify a name
   /// filter and a specific state to filter the vocabularies list.
   /// </summary>
   /// <param name="nameContains">Optional string the vocabulary name must
contain.</param>
   /// <param name="stateEquals">Optional state of the vocabulary.</param>
   /// <returns>List of information about the vocabularies.</returns>
   public async Task<List<VocabularyInfo>> ListCustomVocabularies(string?
nameContains = null,
       VocabularyState? stateEquals = null)
   {
       var response = await _amazonTranscribeService.ListVocabulariesAsync(
           new ListVocabulariesRequest()
           {
               NameContains = nameContains,
               StateEquals = stateEquals
           });
       return response.Vocabularies;
   }
```

- API の詳細については、「 API リファレンスListVocabularies」の「」を参照してください。AWS SDK for .NET

StartMedicalTranscriptionJob

次の例は、StartMedicalTranscriptionJob を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Start a medical transcription job for a media file. This method returns
    /// as soon as the job is started.
    /// </summary>
    /// <param name="jobName">A unique name for the medical transcription job.</
param>
    /// <param name="mediaFileUri">The URI of the media file, typically an Amazon S3
 location.</param>
    /// <param name="mediaFormat">The format of the media file.</param>
    /// <param name="outputBucketName">Location for the output, typically an Amazon
 S3 location.</param>
    /// <param name="transcriptionType">Conversation or dictation transcription
 type.</param>
    /// <returns>A MedicalTransactionJob instance with information on the new job.</
returns>
    public async Task<MedicalTranscriptionJob> StartMedicalTranscriptionJob(
        string jobName, string mediaFileUri,
        MediaFormat mediaFormat, string outputBucketName,
 Amazon.TranscribeService.Type transcriptionType)
    {
        var response = await
 _amazonTranscribeService.StartMedicalTranscriptionJobAsync(
            new StartMedicalTranscriptionJobRequest()
            {
                MedicalTranscriptionJobName = jobName,
                Media = new Media()
                {
                    MediaFileUri = mediaFileUri
                },
                MediaFormat = mediaFormat,
```

```
                LanguageCode =
                    LanguageCode
                        .EnUS, // The value must be en-US for medical
transcriptions.
                OutputBucketName = outputBucketName,
                OutputKey =
                    jobName, // The value is a key used to fetch the output of the
transcription.
                Specialty = Specialty.PRIMARYCARE, // The value PRIMARYCARE must be
set.
                Type = transcriptionType
            });
        return response.MedicalTranscriptionJob;
    }
```

- API の詳細については、「 API リファレンスStartMedicalTranscriptionJob」の「」を参照してください。 AWS SDK for .NET

`StartTranscriptionJob`

次の例は、StartTranscriptionJob を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Start a transcription job for a media file. This method returns
    /// as soon as the job is started.
    /// </summary>
    /// <param name="jobName">A unique name for the transcription job.</param>
    /// <param name="mediaFileUri">The URI of the media file, typically an Amazon S3
location.</param>
    /// <param name="mediaFormat">The format of the media file.</param>
```

```
    /// <param name="languageCode">The language code of the media file, such as en-
US.</param>
    /// <param name="vocabularyName">Optional name of a custom vocabulary.</param>
    /// <returns>A TranscriptionJob instance with information on the new job.</
returns>
    public async Task<TranscriptionJob> StartTranscriptionJob(string jobName, string
 mediaFileUri,
        MediaFormat mediaFormat, LanguageCode languageCode, string? vocabularyName)
    {
        var response = await _amazonTranscribeService.StartTranscriptionJobAsync(
            new StartTranscriptionJobRequest()
            {
                TranscriptionJobName = jobName,
                Media = new Media()
                {
                    MediaFileUri = mediaFileUri
                },
                MediaFormat = mediaFormat,
                LanguageCode = languageCode,
                Settings = vocabularyName != null ? new Settings()
                {
                    VocabularyName = vocabularyName
                } : null
            });
        return response.TranscriptionJob;
    }
```

- API の詳細については、「 API リファレンスStartTranscriptionJob」の「」を参照してください。 AWS SDK for .NET

UpdateVocabulary

次の例は、UpdateVocabulary を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Update a custom vocabulary with new values. Update overwrites all existing
information.
    /// </summary>
    /// <param name="languageCode">The language code of the vocabulary.</param>
    /// <param name="phrases">Phrases to use in the vocabulary.</param>
    /// <param name="vocabularyName">Name for the vocabulary.</param>
    /// <returns>The state of the custom vocabulary.</returns>
    public async Task<VocabularyState> UpdateCustomVocabulary(LanguageCode
languageCode,
        List<string> phrases, string vocabularyName)
    {
        var response = await _amazonTranscribeService.UpdateVocabularyAsync(
            new UpdateVocabularyRequest()
            {
                LanguageCode = languageCode,
                Phrases = phrases,
                VocabularyName = vocabularyName
            });
        return response.VocabularyState;
    }
```

- API の詳細については、「 API リファレンスUpdateVocabulary」の「」を参照してください。AWS SDK for .NET

を使用した Amazon Translate の例 AWS SDK for .NET

次のコード例は、Amazon Translate AWS SDK for .NET で 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、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、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、 へのリンクが含まれています。このリンクには GitHub、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が記載されています。

トピック

- アクション

アクション

`DescribeTextTranslationJob`

次の例は、`DescribeTextTranslationJob` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

/// <summary>
/// The following example shows how to retrieve the details of
/// a text translation job using Amazon Translate.
/// </summary>
public class DescribeTextTranslation
{
    public static async Task Main()
    {
        var client = new AmazonTranslateClient();
```

```
            // The Job Id is generated when the text translation job is started
            // with a call to the StartTextTranslationJob method.
            var jobId = "1234567890abcdef01234567890abcde";

            var request = new DescribeTextTranslationJobRequest
            {
                JobId = jobId,
            };

            var jobProperties = await DescribeTranslationJobAsync(client, request);

            DisplayTranslationJobDetails(jobProperties);
        }

        /// <summary>
        /// Retrieve information about an Amazon Translate text translation job.
        /// </summary>
        /// <param name="client">The initialized Amazon Translate client object.</
param>
        /// <param name="request">The DescribeTextTranslationJobRequest object.</
param>
        /// <returns>The TextTranslationJobProperties object containing
        /// information about the text translation job..</returns>
        public static async Task<TextTranslationJobProperties>
 DescribeTranslationJobAsync(
            AmazonTranslateClient client,
            DescribeTextTranslationJobRequest request)
        {
            var response = await client.DescribeTextTranslationJobAsync(request);
            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                return response.TextTranslationJobProperties;
            }
            else
            {
                return null;
            }
        }

        /// <summary>
        /// Displays the properties of the text translation job.
        /// </summary>
        /// <param name="jobProperties">The properties of the text translation
```

```
        /// job returned by the call to DescribeTextTranslationJobAsync.</param>
        public static void DisplayTranslationJobDetails(TextTranslationJobProperties
jobProperties)
        {
            if (jobProperties is null)
            {
                Console.WriteLine("No text translation job properties found.");
                return;
            }

            // Display the details of the text translation job.
            Console.WriteLine($"{jobProperties.JobId}: {jobProperties.JobName}");
        }
    }
```

- API の詳細については、「 API リファレンスDescribeTextTranslationJob」の「」を参照してください。 AWS SDK for .NET

`ListTextTranslationJobs`

次の例は、`ListTextTranslationJobs` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon.Translate;
    using Amazon.Translate.Model;

    /// <summary>
    /// List Amazon Translate translation jobs, along with details about each job.
    /// </summary>
```

```
    public class ListTranslationJobs
    {
        public static async Task Main()
        {
            var client = new AmazonTranslateClient();
            var filter = new TextTranslationJobFilter
            {
                JobStatus = "COMPLETED",
            };

            var request = new ListTextTranslationJobsRequest
            {
                MaxResults = 10,
                Filter = filter,
            };

            await ListJobsAsync(client, request);
        }

        /// <summary>
        /// List Amazon Translate text translation jobs.
        /// </summary>
        /// <param name="client">The initialized Amazon Translate client object.</
param>
        /// <param name="request">An Amazon Translate
        /// ListTextTranslationJobsRequest object detailing which text
        /// translation jobs are of interest.</param>
        public static async Task ListJobsAsync(
            AmazonTranslateClient client,
            ListTextTranslationJobsRequest request)
        {
            ListTextTranslationJobsResponse response;

            do
            {
                response = await client.ListTextTranslationJobsAsync(request);

 ShowTranslationJobDetails(response.TextTranslationJobPropertiesList);

                request.NextToken = response.NextToken;
            }
            while (response.NextToken is not null);
        }
```

```
        /// <summary>
        /// List existing translation job details.
        /// </summary>
        /// <param name="properties">A list of Amazon Translate text
        /// translation jobs.</param>
        public static void
ShowTranslationJobDetails(List<TextTranslationJobProperties> properties)
        {
            properties.ForEach(prop =>
            {
                Console.WriteLine($"{prop.JobId}: {prop.JobName}");
                Console.WriteLine($"Status: {prop.JobStatus}");
                Console.WriteLine($"Submitted time: {prop.SubmittedTime}");
            });
        }
    }
```

- API の詳細については、「 API リファレンスListTextTranslationJobs」の「」を参照してください。 AWS SDK for .NET

`StartTextTranslationJob`

次の例は、`StartTextTranslationJob` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon.Translate;
    using Amazon.Translate.Model;

    /// <summary>
```

```
    /// This example shows how to use Amazon Translate to process the files in
    /// an Amazon Simple Storage Service (Amazon S3) bucket. The translated results
    /// will also be stored in an Amazon S3 bucket.
    /// </summary>
    public class BatchTranslate
    {
        public static async Task Main()
        {
            var contentType = "text/plain";

            // Set this variable to an S3 bucket location with a folder."
            // Input files must be in a folder and not at the bucket root."
            var s3InputUri = "s3://DOC-EXAMPLE-BUCKET1/FOLDER/";
            var s3OutputUri = "s3://DOC-EXAMPLE-BUCKET2/";

            // This role must have permissions to read the source bucket and to read
 and
            // write to the destination bucket where the translated text will be
 stored.
            var dataAccessRoleArn = "arn:aws:iam::0123456789ab:role/
S3TranslateRole";

            var client = new AmazonTranslateClient();

            var inputConfig = new InputDataConfig
            {
                ContentType = contentType,
                S3Uri = s3InputUri,
            };

            var outputConfig = new OutputDataConfig
            {
                S3Uri = s3OutputUri,
            };

            var request = new StartTextTranslationJobRequest
            {
                JobName = "ExampleTranslationJob",
                DataAccessRoleArn = dataAccessRoleArn,
                InputDataConfig = inputConfig,
                OutputDataConfig = outputConfig,
                SourceLanguageCode = "en",
                TargetLanguageCodes = new List<string> { "fr" },
            };
```

```
            var response = await StartTextTranslationAsync(client, request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"{response.JobId}: {response.JobStatus}");
            }
        }

        /// <summary>
        /// Start the Amazon Translate text translation job.
        /// </summary>
        /// <param name="client">The initialized AmazonTranslateClient object.</
param>
        /// <param name="request">The request object that includes details such
        /// as source and destination bucket names and the IAM Role that will
        /// be used to access the buckets.</param>
        /// <returns>The StartTextTranslationResponse object that includes the
        /// details of the request response.</returns>
        public static async Task<StartTextTranslationJobResponse>
 StartTextTranslationAsync(AmazonTranslateClient client,
 StartTextTranslationJobRequest request)
        {
            var response = await client.StartTextTranslationJobAsync(request);
            return response;
        }
    }
```

- API の詳細については、「 API リファレンスStartTextTranslationJob」の「」を参照してください。 AWS SDK for .NET

StopTextTranslationJob

次の例は、StopTextTranslationJob を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

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

/// <summary>
/// Shows how to stop a running Amazon Translation Service text translation
/// job.
/// </summary>
public class StopTextTranslationJob
{
    public static async Task Main()
    {
        var client = new AmazonTranslateClient();
        var jobId = "1234567890abcdef01234567890abcde";

        var request = new StopTextTranslationJobRequest
        {
            JobId = jobId,
        };

        await StopTranslationJobAsync(client, request);
    }

    /// <summary>
    /// Sends a request to stop a text translation job.
    /// </summary>
    /// <param name="client">Initialized AmazonTrnslateClient object.</param>
    /// <param name="request">The request object to be passed to the
    /// StopTextJobAsync method.</param>
    public static async Task StopTranslationJobAsync(
        AmazonTranslateClient client,
        StopTextTranslationJobRequest request)
    {
```

```
            var response = await client.StopTextTranslationJobAsync(request);
            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"{response.JobId} as status:
{response.JobStatus}");
            }
        }
    }
```

- API の詳細については、「 API リファレンスStopTextTranslationJob」の「」を参照してください。 AWS SDK for .NET

`TranslateText`

次の例は、`TranslateText` を使用する方法を説明しています。

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
using System;
using System.IO;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Transfer;
using Amazon.Translate;
using Amazon.Translate.Model;

/// <summary>
/// Take text from a file stored a Amazon Simple Storage Service (Amazon S3)
/// object and translate it using the Amazon Transfer Service.
/// </summary>
public class TranslateText
{
    public static async Task Main()
```

```
        {
            // If the region you want to use is different from the region
            // defined for the default user, supply it as a parameter to the
            // Amazon Translate client object constructor.
            var client = new AmazonTranslateClient();

            // Set the source language to "auto" to request Amazon Translate to
            // automatically detect te language of the source text.

            // You can get a list of the languages supposed by Amazon Translate
            // in the Amazon Translate Developer's Guide here:
            //      https://docs.aws.amazon.com/translate/latest/dg/what-is.html
            string srcLang = "en"; // English.
            string destLang = "fr"; // French.

            // The Amazon Simple Storage Service (Amazon S3) bucket where the
            // source text file is stored.
            string srcBucket = "DOC-EXAMPLE-BUCKET";
            string srcTextFile = "source.txt";

            var srcText = await GetSourceTextAsync(srcBucket, srcTextFile);
            var destText = await TranslatingTextAsync(client, srcLang, destLang,
srcText);

            ShowText(srcText, destText);
        }

        /// <summary>
        /// Use the Amazon S3 TransferUtility to retrieve the text to translate
        /// from an object in an S3 bucket.
        /// </summary>
        /// <param name="srcBucket">The name of the S3 bucket where the
        /// text is stored.
        /// </param>
        /// <param name="srcTextFile">The key of the S3 object that
        /// contains the text to translate.</param>
        /// <returns>A string representing the source text.</returns>
        public static async Task<string> GetSourceTextAsync(string srcBucket, string
srcTextFile)
        {
            string srcText = string.Empty;

            var s3Client = new AmazonS3Client();
            TransferUtility utility = new TransferUtility(s3Client);
```

```
            using var stream = await utility.OpenStreamAsync(srcBucket,
srcTextFile);

            StreamReader file = new System.IO.StreamReader(stream);

            srcText = file.ReadToEnd();
            return srcText;
        }

        /// <summary>
        /// Use the Amazon Translate Service to translate the document from the
        /// source language to the specified destination language.
        /// </summary>
        /// <param name="client">The Amazon Translate Service client used to
        /// perform the translation.</param>
        /// <param name="srcLang">The language of the source text.</param>
        /// <param name="destLang">The destination language for the translated
        /// text.</param>
        /// <param name="text">A string representing the text to ranslate.</param>
        /// <returns>The text that has been translated to the destination
        /// language.</returns>
        public static async Task<string> TranslatingTextAsync(AmazonTranslateClient
client, string srcLang, string destLang, string text)
        {
            var request = new TranslateTextRequest
            {
                SourceLanguageCode = srcLang,
                TargetLanguageCode = destLang,
                Text = text,
            };

            var response = await client.TranslateTextAsync(request);

            return response.TranslatedText;
        }

        /// <summary>
        /// Show the original text followed by the translated text.
        /// </summary>
        /// <param name="srcText">The original text to be translated.</param>
        /// <param name="destText">The translated text.</param>
        public static void ShowText(string srcText, string destText)
        {
```

```
            Console.WriteLine("Source text:");
            Console.WriteLine(srcText);
            Console.WriteLine();
            Console.WriteLine("Translated text:");
            Console.WriteLine(destText);
        }
    }
```

- API の詳細については、「 API リファレンスTranslateText」の「」を参照してください。 AWS SDK for .NET

を使用したクロスサービスの例 AWS SDK for .NET

以下のサンプルアプリケーションでは、 AWS SDK for .NET を使用して複数の AWS のサービスアプリケーションで動作します。

クロスサービスの例は、アプリケーションの構築を始めるのに役立つ上級レベルの経験を対象としています。

例

- メッセージを翻訳する公開およびサブスクリプションアプリケーションを構築する
- ユーザーがラベルを使用して写真を管理できる写真アセット管理アプリケーションの作成
- DynamoDB データを追跡するウェブアプリケーションを作成する
- Aurora Serverless 作業項目トラッカーの作成
- 顧客からのフィードバックを分析し、音声を合成するアプリケーションの作成
- AWS SDK を使用して Amazon Rekognition でイメージ内のオブジェクトを検出する
- S3 Object Lambda でアプリケーションのデータを変換する
- Message AWS Processing Framework for .NET を使用して Amazon SQS メッセージを公開および受信する

メッセージを翻訳する公開およびサブスクリプションアプリケーションを構築する

AWS SDK for .NET

Amazon Simple Notification Service .NET API を使用して、サブスクリプションおよびパブリッシュ機能を持つウェブアプリケーションを作成する方法を説明します。さらに、このサンプルアプリケーションではメッセージを翻訳します。

完全なソースコードとセットアップと実行の手順については、「」の詳細な例を参照してくださいGitHub。

この例で使用されているサービス

- Amazon SNS
- Amazon Translate

ユーザーがラベルを使用して写真を管理できる写真アセット管理アプリケーションの作成

AWS SDK for .NET

Amazon Rekognition を使用して画像内のラベルを検出し、保存して後で取得できるようにする写真アセット管理アプリケーションの開発方法を示します。

完全なソースコードとセットアップと実行の手順については、「」の詳細な例を参照してください GitHub。

この例のソースについて詳しくは、AWS コミュニティでブログ投稿を参照してください。

この例で使用されているサービス

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

DynamoDB データを追跡するウェブアプリケーションを作成する

AWS SDK for .NET

Amazon DynamoDB .NET API を使用して、DynamoDB 作業データを追跡する動的ウェブアプリケーションを作成する方法を示しています。

完全なソースコードとセットアップと実行の手順については、「」の詳細な例を参照してくださいGitHub。

この例で使用されているサービス

- DynamoDB
- Amazon SES

Aurora Serverless 作業項目トラッカーの作成

AWS SDK for .NET

を使用して、Amazon Aurora データベース内の作業項目を追跡し、Amazon Simple Email Service (Amazon SES) を使用してレポートを E メールで送信するウェブアプリケーション AWS SDK for .NET を作成する方法を示します。この例では、React.js で構築されたフロントエンドを使用して RESTful .NET バックエンドと対話します。

- React ウェブアプリケーションを AWS サービスと統合します。
- Aurora テーブルの項目を一覧表示、更新、削除します。
- Amazon SES を使用して、フィルター処理された作業項目の E メールレポートを送信します。
- 含まれている AWS CloudFormation スクリプトを使用してサンプルリソースをデプロイおよび管理します。

完全なソースコードとセットアップと実行の手順については、「」の詳細な例を参照してくださいGitHub。

この例で使用されているサービス

- Aurora
- Amazon RDS
- Amazon RDS データサービス

- Amazon SES

顧客からのフィードバックを分析し、音声を合成するアプリケーションの作成

AWS SDK for .NET

このサンプルアプリケーションは、顧客フィードバックカードを分析し、保存します。具体的には、ニューヨーク市の架空のホテルのニーズを満たします。このホテルでは、お客様からのフィードバックをさまざまな言語で書かれた実際のコメントカードの形で受け取ります。そのフィードバックは、ウェブクライアントを通じてアプリにアップロードされます。コメントカードの画像をアップロードされると、次の手順が発生します。

- テキストは Amazon Textract を使用して、画像から抽出されます。
- Amazon Comprehend は、抽出されたテキストの感情とその言語を決定します。
- 抽出されたテキストは、Amazon Translate を使用して英語に翻訳されます。
- Amazon Polly は抽出されたテキストからオーディオファイルを合成します。

完全なアプリは AWS CDK を使用してデプロイすることができます。ソースコードとデプロイ手順については、「」の「 GitHubプロジェクト」を参照してください。

この例で使用されているサービス

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

AWS SDK を使用して Amazon Rekognition でイメージ内のオブジェクトを検出する

AWS SDK for .NET

Amazon Simple Storage Service (Amazon S3) バケットにあるイメージ内から、Amazon Rekognition を使用してカテゴリ別にオブジェクトを識別するアプリケーションを、Amazon

Rekognition .NET API を使用して作成する方法を示します。アプリケーションは Amazon Simple Email Service (Amazon SES) を使用して、結果を含む E メール通知を管理者に送信します。

完全なソースコードとセットアップと実行の手順については、「」の詳細な例を参照してくださいGitHub。

この例で使用されているサービス

- Amazon Rekognition
- Amazon S3
- Amazon SES

S3 Object Lambda でアプリケーションのデータを変換する

AWS SDK for .NET

オブジェクトがリクエストしたクライアントまたはアプリケーションのニーズに合うように、標準の S3 GET リクエストにカスタムコードを追加し、S3 から取得したリクエストされたオブジェクトを変更する方法を示しています。

完全なソースコードとセットアップと実行の手順については、「」の詳細な例を参照してくださいGitHub。

この例で使用されているサービス

- Lambda
- Amazon S3

Message AWS Processing Framework for .NET を使用して Amazon SQS メッセージを公開および受信する

AWS SDK for .NET

.NET 用 AWS Message Processing Framework のチュートリアルを提供します。このチュートリアルでは、ユーザーが Amazon SQS メッセージを発行できるようにするウェブアプリケーションと、メッセージを受信するコマンドラインアプリケーションを作成します。

完全なソースコードとセットアップと実行の手順については、 AWS SDK for .NET デベロッパーガイドのチュートリアルと の例を参照してくださいGitHub。

この例で使用されているサービス

- Amazon SQS

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 とユーザーの間で共有される責任です。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。当社のセキュリティ責任は で最優先事項であり 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ています。

クラウドにおけるセキュリティ — お客様の責任は、使用している AWS サービス、およびデータの機密性、組織の要件、適用可能な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 、 AWS コンプライアンスプログラム による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におけるデータ保護
- 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
- Amazon S3 暗号化クライアントの移行

この AWS 製品またはサービスにおけるデータ保護

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。お客様は、このインフラストラクチャでホストされてい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のよくある質問」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記事「AWS 責任共有モデルおよび GDPR」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 Identity Center 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 は必須であり TLS 1.3 がお勧めです。
- 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。
- AWS 暗号化ソリューションと、 内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 などの高度なマネージド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 140-2 検証済みの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) 140-2」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は、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配置し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 名前に使用する自由記述のテキストフィールドやタグに入力したデータは、課金や診断ログに使用される場合があります。外部サーバーへの URL を提供する場合は、そのサーバーへのリクエストを検証するための認証情報を URL に含めないように強くお勧めします。

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 リソースの使用を承認する (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 AWS のサービス 使用できる です。

トピック

対象者

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与えられます。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ことがあります。アクセスの管理方法を理解しておくと、管理者に適切な許可をリクエストするうえで役立ちます。の機能にアクセスできない場合は AWS、AWS ID とアクセスのトラブルシューティング「」または 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。

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 で使用する方法の詳細については AWS、使用している の AWS のサービス 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。として、IAM ユーザーとして AWS アカウントのルートユーザー、または IAM ロールを引き受けて認証 (にサインイン AWS) される必要があります。

ID ソースを介して提供された認証情報を使用して、フェデレーティッド ID AWS として にサインインできます。 AWS IAM Identity Center （ IAM Identity Center) ユーザー、会社のシングルサインオン認証、Google または Facebook の認証情報は、フェデレーティッド ID の例です。フェデレーティッ

ドアイデンティティとしてサインインする場合、IAM ロールを使用して、前もって管理者により ID フェデレーションが設定されています。フェデレーション AWS を使用して にアクセスすると、間接的にロールを引き受けることになります。

ユーザーのタイプに応じて、 AWS Management Console または AWS アクセスポータルにサインインできます。へのサインインの詳細については AWS、「 ユーザーガイド」の「 へのサインイン AWS アカウント方法AWS サインイン 」を参照してください。

AWS プログラムで にアクセスする場合、 は Software Development Kit (SDK) とコマンドラインインターフェイス (CLI) AWS を提供し、認証情報を使用してリクエストに暗号で署名します。 AWS ツールを使用しない場合は、リクエストに自分で署名する必要があります。推奨される方法を使用してリクエストを自分で署名する方法の詳細については、IAM ユーザーガイドの API AWS リクエストの署名を参照してください。

使用する認証方法を問わず、追加セキュリティ情報の提供をリクエストされる場合もあります。例えば、 AWS では、多要素認証 (MFA) を使用してアカウントのセキュリティを向上させることをお勧めします。詳細については、『AWS IAM Identity Center ユーザーガイド』の「Multi-factor authentication」(多要素認証) および『IAM ユーザーガイド』の「AWSにおける多要素認証 (MFA) の使用」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、アカウント内のすべての およびリソースへの AWS のサービス 完全なアクセス権を持つ 1 つのサインインアイデンティティから始めます。この ID は AWS アカウント ルートユーザーと呼ばれ、アカウントの作成に使用した E メールアドレスとパスワードでサインインすることでアクセスできます。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。ルートユーザーの認証情報は保護し、ルートユーザーでしか実行できないタスクを実行するときに使用します。ルートユーザーとしてサインインする必要があるタスクの完全なリストについては、『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。

フェデレーティッドアイデンティティ

ベストプラクティスとして、管理者アクセスを必要とするユーザーを含む人間のユーザーに、一時的な認証情報を使用して にアクセスするための ID プロバイダーとのフェデレーションの使用を要求 AWS のサービス します。

フェデレーティッド ID は、エンタープライズユーザーディレクトリ、ウェブ ID プロバイダー、AWS Directory Service、アイデンティティセンターディレクトリのユーザー、または ID ソースを通じて提供された認証情報 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

ティッド ID が にアクセスすると AWS アカウント、ロールを引き受け、ロールは一時的な認証情報を提供します。

アクセスを一元管理する場合は、 AWS IAM Identity Centerを使用することをお勧めします。IAM Identity Center でユーザーとグループを作成することも、独自の ID ソース内のユーザーとグループのセットに接続して同期して、すべての AWS アカウント とアプリケーションで使用することもできます。IAM Identity Center の詳細については、『AWS IAM Identity Center ユーザーガイド』の「What is IAM Identity Center?」(IAM Identity Center 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、単一のユーザーまたはアプリケーションに対して特定のアクセス許可 AWS アカウント を持つ 内のアイデンティティです。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を保有する IAM ユーザーを作成する代わりに、一時認証情報を使用することをお勧めします。ただし、IAM ユーザーでの長期的な認証情報が必要な特定のユースケースがある場合は、アクセスキーをローテーション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「長期的な認証情報を必要とするユースケースのためにアクセスキーを定期的にローテーションする」を参照してください。

IAM グループは、IAM ユーザーの集団を指定するアイデンティティです。グループとしてサインインすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できます。多数のユーザーグループがある場合、グループを使用することで権限の管理が容易になります。例えば、IAMAdmins という名前のグループを設定して、そのグループに IAM リソースを管理する権限を与えることができます。

ユーザーは、ロールとは異なります。ユーザーは 1 人の人または 1 つのアプリケーションに一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユーザーには永続的な長期の認証情報がありますが、ロールでは一時的な認証情報が提供されます。詳細については、『IAM ユーザーガイド』の「IAM ユーザー (ロールではなく) の作成が適している場合」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 AWS アカウント を持つ 内のアイデンティティです。これは IAM ユーザーに似ていますが、特定のユーザーには関連付けられていません。ロール を切り替える AWS Management Console ことで、 で IAM ロールを一時的に引き受けることができます。ロールを引き受けるには、 または AWS API AWS CLI オペレーションを呼び出すか、カスタム URL を使用します。ロールを使用する方法の詳細については、「IAM ユーザーガイド」の「IAM ロールの使用」を参照してください。

IAM ロールと一時的な認証情報は、次の状況で役立ちます:

- フェデレーションユーザーアクセス – フェデレーティッドアイデンティティに権限を割り当てるには、ロールを作成してそのロールの権限を定義します。フェデレーティッドアイデンティティが認証されると、そのアイデンティティはロールに関連付けられ、ロールで定義されている権限が付与されます。フェデレーションの詳細については、『IAM ユーザーガイド』の「サードパーティーアイデンティティプロバイダー向けロールの作成」を参照してください。IAM アイデンティティセンターを使用する場合、権限セットを設定します。アイデンティティが認証後にアクセスできるものを制御するため、IAM Identity Center は、権限セットを IAM のロールに関連付けます。権限セットの詳細については、『AWS 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。
- 一時的な IAM ユーザー権限 - IAM ユーザーまたはロールは、特定のタスクに対して複数の異なる権限を一時的に IAM ロールで引き受けることができます。
- クロスアカウントアクセス - IAM ロールを使用して、自分のアカウントのリソースにアクセスすることを、別のアカウントの人物 (信頼済みプリンシパル) に許可できます。クロスアカウントアクセス権を付与する主な方法は、ロールを使用することです。ただし、一部の では AWS のサービス、(ロールをプロキシとして使用する代わりに) リソースに直接ポリシーをアタッチできます。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違いについては、『IAM ユーザーガイド』の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。
- クロスサービスアクセス — 一部の は、他の の機能 AWS のサービス を使用します AWS のサービス。例えば、あるサービスで呼び出しを行うと、通常そのサービスによって Amazon EC2 でアプリケーションが実行されたり、Amazon S3 にオブジェクトが保存されたりします。サービスでは、呼び出し元プリンシパルの権限、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使用してこれを行う場合があります。
  - 転送アクセスセッション (FAS) — IAM ユーザーまたはロールを使用して でアクションを実行する場合 AWS、ユーザーはプリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、 を呼び出すプリンシパルのアクセス許可を AWS のサービス、ダウンストリームサービス AWS のサービス へのリクエストリクエストリクエストと組み合わせて使用します。FAS リクエストは、サービスが他の AWS のサービス またはリソースとのやり取りを完了する必要があるリクエストを受け取った場合にのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。
  - サービスロール - サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細

については、IAM ユーザーガイドの「 AWS のサービスにアクセス許可を委任するロールの作成」を参照してください。

- サービスにリンクされたロール – サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント 、サービスによって所有されます。IAM 管理者は、サービスにリンクされたロールの権限を表示できますが、編集することはできません。

- Amazon EC2 で実行されているアプリケーション – IAM ロールを使用して、EC2 インスタンスで実行され、 AWS CLI または AWS API リクエストを行うアプリケーションの一時的な認証情報を管理できます。これは、EC2 インスタンス内でのアクセスキーの保存に推奨されます。 AWS ロールを EC2 インスタンスに割り当て、そのすべてのアプリケーションで使用できるようにするには、インスタンスにアタッチされたインスタンスプロファイルを作成します。インスタンスプロファイルにはロールが含まれ、EC2 インスタンスで実行されるプログラムは一時的な認証情報を取得できます。詳細については、『IAM ユーザーガイド』の「Amazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用して権限を付与する」を参照してください。

IAM ロールと IAM ユーザーのどちらを使用するかについては、『IAM ユーザーガイド』の「(IAM ユーザーではなく) IAM ロールをいつ作成したら良いのか?」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、 AWS ID 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ときにアクセス許可を定義する のオブジェクトです。 は、プリンシパル (ユーザー、ルートユーザー、またはロールセッション) AWS 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。ポリシーでの権限により、リクエストが許可されるか拒否されるかが決まります。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構造と内容の詳細については、「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 AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

デフォルトでは、ユーザーやロールに権限はありません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き継ぐことができます。

IAM ポリシーは、オペレーションの実行方法を問わず、アクションの権限を定義します。例えば、`iam:GetRole` アクションを許可するポリシーがあるとします。そのポリシーを持つユーザーは、 AWS Management Console、、 AWS CLIまたは AWS API からロール情報を取得できます。

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ポリシーは、IAM ユーザー、ユーザーのグループ、ロールなど、アイデンティティにアタッチできる JSON 権限ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースのポリシーを作成する方法については、IAM ユーザーガイドの「IAM ポリシーの作成」を参照してください。

アイデンティティベースポリシーは、さらにインラインポリシーまたはマネージドポリシーに分類できます。インラインポリシーは、単一のユーザー、グループ、またはロールに直接埋め込まれています。管理ポリシーは、 内の複数のユーザー、グループ、ロールにアタッチできるスタンドアロンポリシーです AWS アカウント。管理ポリシーには、 AWS 管理ポリシーとカスタマー管理ポリシーが含まれます。マネージドポリシーまたは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「マネージドポリシーとインラインポリシーの比較」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。リソースベースのポリシーには例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあげられます。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。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。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。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、または を含めることができます AWS のサービス。

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権限を持つかをコントロールします。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、 AWS WAF、および Amazon VPC は、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、一般的ではない追加のポリシータイプをサポートします。これらのポリシータイプでは、より一般的なポリシータイプで付与された最大の権限を設定できます。

- アクセス許可の境界 - アクセス許可の境界は、アイデンティティベースのポリシーによって IAM エンティティ (IAM ユーザーまたはロール) に付与できる権限の上限を設定する高度な機能です。エンティティにアクセス許可の境界を設定できます。結果として得られる権限は、エンティティのアイデンティティベースポリシーとそのアクセス許可の境界の共通部分になります。`Principal` フィールドでユーザーまたはロールを指定するリソースベースのポリシーでは、アクセス許可の境界は制限されません。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。許可の境界の詳細については、「IAM ユーザーガイド」の「IAM エンティティの許可の境界」を参照してください。
- サービスコントロールポリシー (SCPs) – SCPs は、 の組織または組織単位 (OU) に対する最大アクセス許可を指定する JSON ポリシーです AWS Organizations。 AWS Organizations は、 AWS アカウント ビジネスが所有する複数の をグループ化して一元管理するサービスです。組織内のすべての機能を有効にすると、サービスコントロールポリシー (SCP) を一部またはすべてのアカウントに適用できます。SCP は、各 を含むメンバーアカウントのエンティティのアクセス許可を制限します AWS アカウントのルートユーザー。Organizations と SCP の詳細については、『AWS Organizations ユーザーガイド』の「SCP の仕組み」を参照してください。
- セッションポリシー - セッションポリシーは、ロールまたはフェデレーションユーザーの一時的なセッションをプログラムで作成する際にパラメータとして渡す高度なポリシーです。結果としてセッションの権限は、ユーザーまたはロールのアイデンティティベースポリシーとセッションポリシーの共通部分になります。また、リソースベースのポリシーから権限が派生する場合もあります。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。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権限を理解するのがさらに難しくなります。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 の AWS のサービス 仕組み

ほとんどの IAM 機能との AWS のサービス 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、 と 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 と修正に役立ちます。

トピック

- でアクションを実行する権限がない AWS
- iam を実行する権限がありません。PassRole
- 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ない AWS

「I am not authorized to perform an action in Amazon Bedrock」というエラーが表示された場合、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。サインイン資格情報を提供した担当者が管理者です。

iam を実行する権限がありません。PassRole

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権限がありません。

```
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 の AWS のサービス 仕組み。
- 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) へのアクセス権限」を参照してください。
- クロスアカウントアクセスでのロールとリソースベースのポリシーの使用の違いの詳細については、「IAM ユーザーガイド」の「IAM ロールとリソースベースのポリシーとの相違点」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、コンプライアンスプログラムAWS のサービス による対象範囲内のコンプライアンスプログラムを参照し、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、「 でのレポートのダウンロード AWS Artifact」の」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。 は、コンプライアンスに役立つ以下のリソース AWS を提供しています。

- セキュリティとコンプライアンスのクイックスタートガイド – これらのデプロイガイドでは、アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスに重点を置いたベースライン環境 AWS を にデプロイする手順について説明します。
- アマゾン ウェブ サービスにおける HIPAA セキュリティとコンプライアンスのアーキテクチャ — このホワイトペーパーでは、企業が AWS を使用して HIPAA 対象アプリケーションを作成する方法について説明します。

> **Note**
>
> すべて AWS のサービス HIPAA の対象となるわけではありません。詳細については、「HIPAA 対応サービスのリファレンス」を参照してください。

- AWS コンプライアンスリソース – このワークブックとガイドのコレクションは、お客様の業界や地域に適用される場合があります。
- AWS カスタマーコンプライアンスガイド — コンプライアンスの観点から責任共有モデルを理解します。このガイドでは、ガイダンスを保護し AWS のサービス 、複数のフレームワーク (米国国立標準技術研究所 (NIST)、Payment Card Industry Security Standards Council (PCI)、国際標準化

機構 (ISO) を含む) のセキュリティコントロールにマッピングするためのベストプラクティスをまとめています。

- 「 デベロッパーガイド」の「 ルールによるリソースの評価」 – この AWS Config サービスは、リソース設定が社内プラクティス、業界ガイドライン、および規制にどの程度準拠しているかを評価します。 AWS Config
- AWS Security Hub – これにより AWS のサービス 、 内のセキュリティ状態を包括的に確認できます AWS。Security Hub では、セキュリティコントロールを使用して AWS リソースを評価し、セキュリティ業界標準とベストプラクティスに対するコンプライアンスをチェックします。サポートされているサービスとコントロールのリストについては、「Security Hub のコントロールリファレンス」を参照してください。
- Amazon GuardDuty – これにより AWS アカウント、疑わしいアクティビティや悪意のあるアクティビティがないか環境を監視することで、、、ワークロード、コンテナ、データに対する潜在的な脅威 AWS のサービス を検出します。 GuardDuty は、特定のコンプライアンスフレームワークで義務付けられている侵入検知要件を満たすことで、PCI DSS などのさまざまなコンプライアンス要件への対応に役立ちます。
- AWS Audit Manager – これにより AWS のサービス 、 AWS 使用状況を継続的に監査し、リスクの管理方法と規制や業界標準への準拠を簡素化できます。

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 、 AWS コンプライアンスプログラム による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、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た、物理的に分離および隔離された複数のアベイラビリティーゾーンを提供します。

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イ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が高く、フォールトトレラントで、スケーラブルです。

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 、 AWS コンプライアンスプログラム による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保護されています AWS 。 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。

が AWS 公開した 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。クライアントは以下をサポートする必要があります:

- Transport Layer Security (TLS)。TLS 1.2 は必須で TLS 1.3 がお勧めです。
- 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 、 AWS コンプライアンスプログラム による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 で実行している場合は、https://learn.microsoft.com/en-us/windows-server/security/tls/tls-registry-settings#tls-12 で説明されているように、レジストリで TLS 1.2 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。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 日以降、 AWS SDK for .NET は .NET Framework 3.5 のサポートを終了し、.NET Framework の最小バージョンを 4.6.2 に変更します。詳細については、ブログ記事「 の .NET Framework 3.5 および 4.5 ターゲットに予定されている重要な変更点 AWS SDK for .NET」を参照してください。

アプリケーションが Windows 7 SP1 または Windows Server 2008 R2 SP1 の新しい .NET Framework で実行されている場合は、https://learn.microsoft.com/en-us/windows-server/security/tls/

tls-registry-settings#tls-12 で説明されているように、レジストリで TLS 1.2 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。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 は、 AWS サービスへのすべての呼び出し AWS SDK for .NET に使用します。環境の動作は、次のように、実行中の Windows PowerShell のバージョンによって異なります。

Windows PowerShell 2.0 から 5.x

Windows PowerShell 2.0 から 5.x は .NET Framework で実行されます。次のコマンド PowerShell を使用して、 が使用している .NET ランタイム (2.0 または 4.0) を確認できます。

```
$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 日以降、 AWS SDK for .NET は .NET Framework 3.5 のサポートを終了し、.NET Framework の最小バージョンを 4.6.2 に変更します。詳細については、ブログ記事「 の .NET Framework 3.5 および 4.5 ターゲットに予定されている重要な変更点 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 で実行されます。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/en-us/xamarin/cross-platform/app-fundamentals/transport-layer-security の指示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。
- プロジェクトプロパティ 、Android オプション： HttpClient 実装を Android に設定し、SSL/TLS 実装をネイティブ TLS 1.2 以上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。
- プロジェクトプロパティ 、iOS ビルド : 実装は NSUrlSession に設定する必要があります。 HttpClient

macOS の場合

- macOS 10.9 以降が必要です。
- プロジェクトオプション 、ビルド 、Mac ビルド ： HttpClient 実装は 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。これは、https://docs.unity3d.com/2019.1/Documentation/Manual/ScriptingRuntimeUpgrade.html で説明されているように、プロジェクト設定 、設定 、プレイヤー で設定できます。.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。詳細については、https://blog.unity.com/technology/scripting-runtime-improvements-in-unity-2018-2 を参照してください。

ブラウザ (Blazor 用 WebAssembly)

WebAssembly はサーバー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 は WebAssembly、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。をサポートする前に、すべてのメインストリームブラウザが TLS 1.2 をサポートしていました WebAssembly。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。

Amazon S3 暗号化クライアントの移行

このトピックでは、アプリケーションで使用している Amazon Simple Storage Service (Amazon S3) 暗号化クライアントをバージョン 1 (V1) からバージョン 2 (V2) に移行し、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 AWSSDKKeyManagementService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するか、更新したバージョンがこの新しいパッケージで取り込まれるように削除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);
```

AWS SDK for .NET に関する特別な考慮事項

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

AWS SDK for .NET 用にアセンブリを入手する

このトピックでは、 AWSSDK アセンブリを入手してプロジェクトで使用するようにローカル (またはオンプレミス) に保存する方法を説明します。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、各プロジェクトに必要なパッケージのみ NuGetをダウンロードしてインストールすることをお勧めします。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およびインストールできない、または許可されていない場合は、次のオプションを使用できます。

ZIP ファイルをダウンロードして抽出する

(この方法は、AWS SDK for .NET のリファレンスを処理するために推奨される方法ではありません。)

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 Standard 2.0 および 2.1 をサポートするアセンブリ。
- aws-sdk-net45.zip - .NET Framework 4.5 以降をサポートするアセンブリ。
- aws-sdk-net35.zip - .NET Framework 3.5 をサポートするアセンブリ。

> **Warning**
>
> 2024 年 8 月 15 日以降、 AWS SDK for .NETは .NET Framework 3.5 のサポートを終了し、.NET Framework の最小バージョンを 4.6.2 に変更します。詳細については、ブログ記事「 の .NET Framework 3.5 および 4.5 ターゲットに関する重要な変更点AWS SDK for .NET」を参照してください。

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場合、「認証情報とプロファイルの解決」の説明に従って AWS SDK for .NET が認証情報を検索して取得できるようにする手法が推奨されます。

ただし、プロファイルと認証情報を能動的に取得するようアプリケーションを設定して、AWS のサービスクライアント作成時にそれらの認証情報を明示的に使用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を使用しているファイル (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) 内で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、(次で説明する) `AWSCredentialsFactory` クラスを使用する代わりにこのクラスを使用して、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 または空の場合は、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。
- `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 AWS 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のプロファイルの取得

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、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しているファイルから取得しますが、これは別の場所にあります。

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 AWS 認証情報を作成し、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。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 を使用する際、アプリケーションは、NuGet を使用せず、AWS SDK for .NET アセンブリ (DLL ファイル) を直接参照します。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て、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 アセンブリの取得」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。

- AWSSDK.Core および使用する AWS サービスの DLL と同時に、Unity プロジェクトに次の DLL を含める必要があります。バージョン 3.5.109 の AWS SDK for .NET から開始すると、.NET Standard の ZIP ファイルにこれらの追加 DLL 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重要な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。

これらの特別な考慮事項に加えて、バージョン 3.5 の AWS SDK for .NET への Unity アプリケーションの移行については、バージョン 3.5 の変更点 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。

AWS SDK for .NET 用の API リファレンス。

AWS SDK for .NET は、AWS サービスにアクセスするために使用する API を提供します。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、ガイダンス付きのコード例 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。

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	Message AWS Processing Framework for .NET のプレビューリリースに関する情報を追加	2024 年 3 月 28 日
最新情報	.NET 8 のサポートに関する情報が含まれました。	2024 年 2 月 23 日
最新情報	.NET Framework サポートへの今後の変更に関する情報を追加しました。	2024 年 2 月 18 日
AWSSDK アセンブリの取得	.NET 8 以降をサポートするアセンブリに関する情報を含めました。	2024 年 1 月 8 日
AWS .NET 用の Message Processing Framework	Message Processing Framework のベータリリースに関する情報が含まれました。	2023 年 12 月 10 日
AWS OpsWorks	のサポート終了に関する注意事項を追加しました AWS OpsWorks。	2023 年 12 月 8 日
Amazon DynamoDB NoSQL データベースの使用	ドキュメントとオブジェクト永続性プログラミングモデルに関する情報が更新されました。コールドスタートやスレッドプールの動作に起因する特定のレイテンシーやデッ	2023 年 11 月 15 日

	ドロック状態を防ぐことができるようになりました。	
IAM ベストプラクティスの更新をさらに追加	IAM ベストプラクティスに沿ってガイドを更新しました。詳細については、「IAM 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。	2023 年 10 月 5 日
AWSSDK アセンブリの取得	廃止された AWS Tools for Windows インストーラ (つまり MSI) AWS SDK for .NET を使用した のインストールに関する情報を削除しました。	2023 年 9 月 25 日
IAM ベストプラクティスの更新	IAM ベストプラクティスに沿ってガイドを更新しました。詳細については、「IAM 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。	2023 年 7 月 18 日
Lambda Annotations	AWS Lambda Annotations フレームワークは一般提供のためにリリースされました。	2023 年 7 月 17 日
最新情報	DynamoDB の分散キャッシュプロバイダのプレビューリリースに関する情報を追加しました。	2023 年 7 月 15 日
目次	目次が更新され、コード例が見つけやすくなりました。	2023 年 6 月 8 日
リージョン解像度	不足しているリージョン仕様を SDK が解決する方法についての情報を追加しました。	2023 年 3 月 14 日

MSI のSupport	AWS Tools for Windows インストーラのサポート終了に関する注意事項を追加しました。	2023 年 3 月 6 日
Lambda Annotations (プレビュー)	AWS Lambda Annotations フレームワークのプレビュー。	2022 年 9 月 22 日
アプリケーションを にデプロイする AWS	メインコンテンツを GitHub ページサイトに移動しました: https://aws.github.io/aws-dotnet-deploy/	2022 年 6 月 28 日
EC2-Classic の廃止	EC2-Classic の廃止に関する注記を追加しました。	2022 年 4 月 13 日
を使用したシングルサインオン AWS SDK for .NET	AWS SDK for .NETを使用する際のシングルサインオン(SSO)に関する情報を追加しました。	2022 年 3 月 17 日
最小 TLS バージョンの適用	TLS 1.3 に関する情報を追加しました。	2022 年 3 月 16 日
AWS サービスの使用	で使用できるコード例のリストが含まれています GitHub。	2022 年 2 月 28 日
SDK メトリクスの有効化	廃止されたSDK メトリクスの有効化についての情報を削除しました。	2022 年 1 月 20 日
アプリケーションを にデプロイする AWS	AWS Toolkit for Visual Studio への参照を追加しました。これにより、デプロイツール と同様の AWS デプロイ機能が提供されます。	2021 年 10 月 26 日

AWS SDK for .NET バージョン 3 ガイドの統合	AWS SDK for .NET バージョン 3 デベロッパーガイド「V3」と「最新」の 2 つが、「v3」URL の下の 1 つのガイドに統合されました。	2021 年 8 月 18 日
.NET Standard 1.3 からの移行	での .NET Standard 1.3 のサポート AWS SDK for .NET は終了します。	2021 年 3 月 25 日
アプリケーションを にデプロイする AWS （プレビュー）	.NET CLI からアプリケーションをデプロイするために使用できる AWS デプロイツールに関するプレビュー情報を追加しました。	2021 年 3 月 15 日
のバージョン 3.5 AWS SDK for .NET	のバージョン 3.5 AWS SDK for .NET がリリースされました。	2020 年 8 月 25 日
ページネーター	多くのサービスクライアントにページネーターが追加され、API の結果のページ分割がより便利になりました。	2020 年 8 月 24 日
再試行とタイムアウト	再試行モードに関する情報を追加しました。	2020 年 8 月 20 日
S3 暗号化クライアントの移行	Amazon S3 暗号化クライアントを V1 から V2 に移行する方法についての情報が追加されました。	2020 年 8 月 7 日
S3 暗号化用 KMS キーの使用	S3 暗号化クライアントのバージョン 2 の使用例を更新しました。	2020 年 8 月 6 日

.NET Standard 1.3 からの移行	2020 年末での .NET Standard 1.3 のサポート終了に関する情報を追加しました。	2020 年 5 月 18 日
クイックスタート	AWS SDK for .NETについて紹介するため、基本的なセットアップとチュートリアルを含むクイックスタートセクションが追加されました。	2020 年 3 月 27 日
TLS 1.2 の適用	SDK で TLS 1.2 を適用する方法についての情報を追加しました。	2020 年 3 月 10 日

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。