AWS SDK for .NET

デベロッパーガイド

Table of Contents

.NET アプリケーションをにデプロイしますかAWS数回クリックするだけで？ 新しい製品をお試しください.NET CLI ツールシンプルなデプロイエクスペリエンスを実現します。Ourブログ投稿でフィードバックを送信してくださいGitHub!

詳細については、ののののセクションを参照してください。デプロイツールこのガイドの「」を参照してください。

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

AWS SDK for .NET とは

-AWS SDK for .NET費用効果の高い、スケーラブルで信頼できる .NET アプリケーションを構築できます。AWSAmazon Simple Storage Service (Amazon S3) や Amazon Elastic Compute Cloud (Amazon EC2) などのサービス。SDK は、AWS.NET 開発者にとって一貫性のある使い慣れた一連のライブラリを提供することにより、サービス。

(わかりました。の準備が整いましたチュートリアル (p. 5)を起動するにはのセットアップ (p. 15)。)

のバージョンについて

Note

このドキュメントは、バージョン 3.0 以降のAWS SDK for .NET。主に.NET CoreとASP.NET Coreを中心としているが、.NET FrameworkやASP.NET 4に関する情報も含まれている。x。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、「」を参照してください。プロジェクトを移行する (p. 62)。
の以前のバージョンで推奨されていないコンテンツを検索するにはAWS SDK for .NETについては、次の項目を参照してください。

- AWS SDK for .NET(バージョン 2、非推奨) 開発者ガイド

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 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

一般的ユースケース

-AWS SDK for .NETを使用すると、以下のようないくつかの説得力のあるユースケースを実現できます。

- ユーザーおよびロールを管理するAWS Identity and Access Management(IAM)。
- へのアクセスAmazon Simple Storage Service (Amazon S3)バケットを作成し、オブジェクトを格納します。
- 管理Amazon Simple Notification Service (Amazon SNS)トピックへの HTTP サブスクリプション。
- を使用するS3 転送ユーティリティを使用して Xamin アプリケーションから Amazon S3 にファイルを転送します。

- を使用するAmazon Simple Queue Service (Amazon SQS)システム内のコンポーネント間でメッセージとワークフローを処理します。
- SQL ステートメントをに送信して、効率的な Amazon S3 転送を実行するAmazon S3 Select。
- の作成と起動Amazon EC2インスタンス、および Amazon EC2 の設定とリクエストスポットインスタンス。

このセクションのその他のトピック

- AWSの関連ツールAWS SDK for .NET (p. 2)
- その他のリソース (p. 3)

AWSの関連ツールAWS SDK for .NET

Windows PowerShell 用ツールと PowerShell Core 用ツール

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。-AWSPowerShell Tools を使用すると、AWSリソースを PowerShell プロンプトから入力します。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 スキーマの使用。
- Amazon ECS タスク定義ファイルを操作するには、IntelliSense を使用します。
- AWS Cloud Development Kit (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、アマゾン ウェブ サービスを使用した .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。♪AWSAmazon 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、「」のセットアップAWS Toolkit for Visual Studio。

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 で作業できます。AWS Elastic Beanstalk,AWS CodeDeploy, Lambda,AWS CloudFormation、Amazon Simple Queue Service (Amazon SQS)、Amazon SNS。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。

開始するには、AWS Toolkit for Azure DevOpsの使用の詳細については、『AWS Toolkit for Microsoft Azure DevOpsユーザーガイド。

その他のリソース

サポートされているサービスとリビジョン履歴

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。

特定のリリースでの変更点については、SDK 変更ログを参照してください。

のホームページ AWS SDK for .NET

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

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検索することができます。徹底したドキュメントと使用例を提供します。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。

AWS フォーラム

にアクセスしてください。AWSフォーラムにアクセスして、質問したり意見や要望を尋ねてください。AWS。各ドキュメントページには、フォーラムへ移動ボタンを使用すると、関連のフォーラムに移動できます。AWSの技術者がフォーラムを見ていて、質問や意見、要望、問題に対応しています。また、いずれのフォーラムについても、RSS フィードにサブスクライブできます。

AWS Toolkit for Visual Studio

Microsoft Visual Studio IDE を使用している場合はをチェックアウトする必要があります。AWS Toolkit for Visual Studioユーザーガイド。

便利なライブラリ、拡張機能、ツール

にアクセスしてください。aws/ドットネットおよびaws/aws-sdk-netGitHub Web サイトの .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。AWS。

このリポジトリの主な内容を以下に示します。

- AWS .NET 構成拡張 (Systems Manager)
- AWS .NET Core セットアップ
- AWS .NET の口グ記録
- Amazon Cognito Authentication 拡張ライブラリ

- AWS X-Ray SDK for .NET

AWS SDK for .NET をすぐに使用開始する

このセクションでは、を初めて使用する開発者向けの基本的なセットアップ手順とチュートリアルが含まれています。AWS SDK for .NET。

詳細については、次のセクションを参照してください。環境をセットアップする (p. 15),プロジェクトをセットアップする (p. 17), およびの使用AWSのサービス (p. 69)

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 (p. 5)
- 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 (p. 10)
- 次のステップ (p. 14)

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 を使用します。AWS SDK for .NET.NET Core は、クロスプラットフォーム開発用です。このチュートリアルでは、SDK を使用して、Amazon S3 バケット自分が所有し、オプションでバケットを作成します。

Steps

- このチュートリアルのセットアップ (p. 5)
- プロジェクトの作成 (p. 7)
- コードの作成 (p. 7)
- アプリケーションを実行する (p. 9)
- クリーンアップ (p. 9)

このチュートリアルのセットアップ

このセクションでは、このチュートリアルを完了するために必要な最小限のセットアップを提供します。これを完全なセットアップであるとは見なさないでください。詳細については、「をセットアップするAWS SDK for .NET環境 (p. 15)」を参照してください。

> Note
>
> 他のチュートリアルまたは既存の設定で次のステップのいずれかをすでに完了している場合は、これらのステップをスキップします。

AWS アカウントを作成する

を作成するにはAWSアカウント、「」を参照してください。Amazon Web Services 新規アカウントを作成してアクティブ化する方法を教えてください。

AWS 認証情報とプロファイルの作成

これらのチュートリアルを実行するには、AWS Identity and Access Management(IAM) ユーザーを作成し、そのユーザーの認証情報を取得します。これらの認証情報を取得したら、開発環境の SDK で利用できるようにします。その方法は次のとおりです。

認証情報を作成して使用するには

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. [Users] を選択して、[Add] を選択します。
3. ユーザー名を指定します。このチュートリアルでは、Dotnet-Tutorial-User を使用します。
4. []SelectAWSアクセスタイプ[] を選択して、プログラムによるアクセス[] を選択してから、次へ: アクセス許可.
5. [Attach existing policies directly (既存のポリシーを直接添付する)] を選択します。
6. Eclipse検索「」と入力します。**s3**[] を選択してから、AmazonS3FullAccess。
7. [Next: (次へ:)] を選択します タグ,次へ: 確認, およびユーザーの作成。
8. Dotnet-Tutorial-User の資格情報を記録します。これを行うには、`.csv` ファイルをダウンロードするか、アクセスキー ID とシークレットアクセスキーをコピーアンドペーストします。

   Warning

   適切なセキュリティ対策を講じてこれらの認証情報を安全に保管し、ローテーションしてください
9. 共有 AWS 認証情報ファイルを作成するか、開きます。このファイルは`~/.aws/credentials`Linux および macOS システムでは、`%USERPROFILE%\.aws\credentials`Windows の場合は。
10. 共有 AWS 認証情報ファイルに次のテキストを追加します。ただし、サンプル ID とサンプルキーを先ほど取得したものに置き換えます。必ずファイルを保存してください。

```
[dotnet-tutorials]
aws_access_key_id = AKIAIOSFODNN7EXAMPLE
aws_secret_access_key = wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

前述の手順は、考えられる認証と認可のうち最もシンプルなものです。詳細については、「」を参照してください。設定AWS認証情報 (p. 20)。

他のツールのインストール

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、「」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 (p. 15)。

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)。通常、これは.NET Core のサポートを含むものとします。

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

プロジェクトの作成

1. コマンドプロンプ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
```

前述のコマンドでは、AWSSDK.S3NuGetパッケージからNuGet パッケージマネージャー。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もよくあります。その場合は、その時点で同様のコマンドを実行できます。

4. 次の一時環境変数を環境に追加します。

Linux または macOS

```
export AWS_PROFILE='dotnet-tutorials'
export AWS_REGION='us-west-2'
```

Windows

```
set AWS_PROFILE=dotnet-tutorials
set AWS_REGION=us-west-2
```

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;

// To interact with Amazon S3.
using Amazon.S3;
using Amazon.S3.Model;

namespace S3CreateAndList
{
  class Program
  {
    // Main method
    static async Task Main(string[] args)
    {
      // Before running this app:
      // - Credentials must be specified in an AWS profile. If you use a profile other
 than
      //   the [default] profile, also set the AWS_PROFILE environment variable.
      // - An AWS Region must be specified either in the [default] profile
      //   or by setting the AWS_REGION environment variable.

      // Create an S3 client object.
```

```
    var s3Client = new AmazonS3Client();

    // 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);
      }
    }

    // List the buckets owned by the user.
    // Call a class method that calls the API method.
    Console.WriteLine("\nGetting a list of your buckets...");
    var listResponse = await MyListBucketsAsync(s3Client);
    Console.WriteLine($"Number of buckets: {listResponse.Buckets.Count}");
    foreach(S3Bucket b in listResponse.Buckets)
    {
      Console.WriteLine(b.BucketName);
    }
  }


  //
  // 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;
  }
```

```
    //
    // Async method to get a list of Amazon S3 buckets.
    private static async Task<ListBucketsResponse> MyListBucketsAsync(IAmazonS3
 s3Client)
    {
      return await s3Client.ListBucketsAsync();
    }

  }
}
```

アプリケーションを実行する

1. 次の 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-quickstart-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。「」の説明に従って、バケット名のルールに従ってください。バケット命名規則のAmazon Simple Storage Service ユーザーガイド。
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、のAmazon S3 コンソールを使用してバケットを削除してください。https://console.aws.amazon.com/s3/。
- このトピックの前のチュートリアルセットアップ中に作成したユーザーを保持しない場合は、のIAM コンソールを使用してユーザーを削除してください。https://console.aws.amazon.com/iam/home#/users。

  ユーザーを削除する場合は、**dotnet-tutorials**共有で作成したプロファイルAWS使用されます。このプロファイルは、このトピックで前述したチュートリアルのセットアップ時に作成しました。
- .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

クイックスタートメニュー (p. 5)に戻るか、このクイックスタートの最後 (p. 14)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 を使用します。AWS SDK for .NETWindows では Visual Studio および .NET Core を使用した。また、SDK を使用して、Amazon S3 バケットバケットを所有し、オプションでバケットを作成すること。

Steps

このチュートリアルのセットアップ

このセクションでは、このチュートリアルを完了するために必要な最小限のセットアップを提供します。これを完全なセットアップであるとは見なさないでください。詳細については、「をセットアップするAWS SDK for .NET環境 (p. 15)」を参照してください。

> Note
>
> 他のチュートリアルまたは既存の設定で次のステップのいずれかをすでに完了している場合は、これらのステップをスキップします。

AWS アカウントを作成する

を作成するにはAWSアカウント、「」を参照してください。Amazon Web Services 新規アカウントを作成してアクティブ化する方法を教えてください。

AWS 認証情報とプロファイルの作成

これらのチュートリアルを実行するには、AWS Identity and Access Management(IAM) ユーザーを作成し、そのユーザーの認証情報を取得します。これらの認証情報を取得したら、開発環境の SDK で利用できるようにします。その方法は次のとおりです。

認証情報を作成して使用するには

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. [Users] を選択して、[Add] を選択します。
3. ユーザー名を指定します。このチュートリアルでは、Dotnet-Tutorial-User を使用します。
4. []SelectAWSアクセスタイプ[] を選択します。プログラムによるアクセス[] を選択してから、次へ: アクセス許可.
5. [Attach existing policies directly (既存のポリシーを直接添付する)] を選択します。
6. Eclipse検索「」と入力します。**s3**[] を選択してから、AmazonS3FullAccess。
7. [Next: (次へ:)] を選択します タグ,次へ: 確認, およびユーザーの作成。
8. Dotnet-Tutorial-User の資格情報を記録します。これを行うには、`.csv` ファイルをダウンロードするか、アクセスキー ID とシークレットアクセスキーをコピーアンドペーストします。

Warning

適切なセキュリティ対策を講じてこれらの認証情報を安全に保管し、ローテーションしてください

9. 共有 AWS 認証情報ファイルを作成するか、開きます。このファイルは`~/.aws/credentials`Linux システムおよび macOS システムでは、`%USERPROFILE%\.aws\credentials`Windows の場合は。
10. 共有 AWS 認証情報ファイルに次のテキストを追加します。ただし、サンプル ID とサンプルキーを先ほど取得したものに置き換えます。必ずファイルを保存してください。

```
[dotnet-tutorials]
aws_access_key_id = AKIAIOSFODNN7EXAMPLE
aws_secret_access_key = wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

前述の手順は、考えられる認証と認可のうち最もシンプルなものです。詳細については、「」を参照してください。設定AWS認証情報 (p. 20)。

他のツールのインストール

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、「」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 (p. 15)。

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。

プロジェクトの作成

1. Visual Studio を開き、C# バージョンの C# を使用する新しいプロジェクトを作成します。コンソールアプリケーションテンプレート。つまり、「... .NET Core 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...」という説明が付いています。プロジェクトに `S3CreateAndList` という名前を付けます。

   Note

   コンソールアプリケーションテンプレートの.NET Framework バージョンを選択しないでください。選択する場合は、必ず .NET Framework 4.5 以降を使用してください。
2. 新しく作成したプロジェクトがロードされた状態で、ツール,NuGet パッケージマネージャー,ソリューション用の NuGet パッケージの管理。
3. AWSSDK.S3 NuGet パッケージを参照し、プロジェクトにインストールします。

   このプロセスでは、AWSSDK.S3からのNuGetパッケージNuGet パッケージマネージャー。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もよくあります。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\netcoreapp3.1` のようになりますが、環境によって異なります。

5. 次の一時環境変数を環境に追加します。

   コマンドプロンプトで、次のコマンドを使用します。

```
set AWS_PROFILE=dotnet-tutorials
set AWS_REGION=us-west-2
```

   または、IDE でアプリケーションを実行する場合は、プロジェクト,S3 プロパティの作成とリスト表示,デバッグそしてそこに設定してください。

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;

// To interact with Amazon S3.
using Amazon.S3;
using Amazon.S3.Model;

namespace S3CreateAndList
{
  class Program
  {
    // Main method
    static async Task Main(string[] args)
    {
      // Before running this app:
      // - Credentials must be specified in an AWS profile. If you use a profile other
 than
      //   the [default] profile, also set the AWS_PROFILE environment variable.
      // - An AWS Region must be specified either in the [default] profile
      //   or by setting the AWS_REGION environment variable.

      // Create an S3 client object.
      var s3Client = new AmazonS3Client();

      // 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);
        }
      }

      // List the buckets owned by the user.
      // Call a class method that calls the API method.
      Console.WriteLine("\nGetting a list of your buckets...");
      var listResponse = await MyListBucketsAsync(s3Client);
```

```
      Console.WriteLine($"Number of buckets: {listResponse.Buckets.Count}");
      foreach(S3Bucket b in listResponse.Buckets)
      {
        Console.WriteLine(b.BucketName);
      }
    }


    //
    // 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;
    }


    //
    // Async method to get a list of Amazon S3 buckets.
    private static async Task<ListBucketsResponse> MyListBucketsAsync(IAmazonS3
 s3Client)
    {
      return await s3Client.ListBucketsAsync();
    }

  }
}
```

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。
   「機能 '非同期メイン」は C# 7.0 では使用できません。言語バージョン 7.1 以上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設定します。これは通常、プロジェクトのプロパティで行われます。をビルドする,アドバンスト。

アプリケーションを実行する

1. コマンドライン引数なしでアプリケーションを実行します。これは、コマンドプロンプト (すでに開いている場合) または IDE から行います。
2. 出力を調べ、所有している Amazon S3 バケットの数 (存在する場合) とその名前を確認します。
3. 新しい Amazon S3 バケットの名前を選択します。「dotnet-quickstart-s3-1-winvs-」をベースとして使用し、GUID や名前などの一意のものを追加します。「」の説明に従って、バケット名のルールに従ってください。バケット命名規則のAmazon Simple Storage Service ユーザーガイド。
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、*BUCKET-NAME*次のコマンドで、選択したバケットの名前を使用します。

   ```
   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,S3 プロパティの作成とリスト表示,デバッグをクリックし、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、Amazon S3 コンソールを使用してバケットを削除してください。https://console.aws.amazon.com/s3/。
- このトピックのチュートリアルのセットアップ中に作成したユーザーを保持しない場合は、のIAM コンソールを使用してユーザーを削除してください。https://console.aws.amazon.com/iam/home#/users。

  ユーザーの削除を選択した場合は、**dotnet-tutorials**共有で作成したプロファイルAWS使用されます。このプロファイルは、このトピックで前述したチュートリアルのセットアップ時に作成しました。
- .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

クイックスタートメニュー (p. 5)に戻るか、このクイックスタートの最後 (p. 14)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リソース (ファイルやフォルダなど) や開発環境内のリソース (ファイルやフォルダなど) を一覧表示します。

AWS SDK for .NET の概要を説明したので、より高度なセットアップ (p. 15)について調べてみましょう。

をセットアップするAWS SDK for .NET環境

このセクションでは、グローバル機能と設定をセットアップする方法を説明します。AWS SDK for .NET。

.NET を初めて使用する場合は、AWSに追加するか、少なくとも新しいAWS SDK for .NETで参照するには、クイックスタート (p. 5)トピックを最初に選択します。SDK の概要を示します。

これらのトピックを終了すると、次はプロジェクトをセットアップする (p. 17)。

トピック

AWS アカウントを作成する

♪AWS SDK for .NETにアクセスするにはAWSのサービスには、AWSアカウントとAWS認証情報。

1. アカウントを作成します。

   を作成するにはAWSアカウント、「」を参照してください。Amazon Web Services 新規アカウントを作成してアクティベートする方法を教えてください。
2. 管理ユーザーを作成します。

   管理コンソールおよびサービスへのアクセスには、root ユーザーアカウント (作成した初期アカウント) を使用しないでください。代わりに、「」で説明されているように、管理ユーザーアカウントを作成します。最初の IAM 管理者のユーザーおよびグループの作成。

   管理ユーザーアカウントを作成してログインの詳細を記録したら、root ユーザーアカウントからサインアウトする管理者アカウントを使用してサインインし直します。

閉じる必要がある場合AWSアカウント、「」を参照してください。アカウントの解約。

次のステップ

ツールチェーンのインストールと設定 (p. 15)

追加情報

証明書およびセキュリティの処理方法に関する詳細は、「」を参照してください。IAM のベストプラクティスとユースケースのIAM ユーザーガイド。

追加の詳細については、AWSアプリケーションの認証情報については、「」を参照してください。設定AWS認証情報 (p. 20)。

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。

Note

SDK のクイックスタート (p. 5)を実行した場合は、これらのツールの一部がすでにインストールされている可能性があります。で .NET 開発を行うのが初めてでもクイックスタートを行っていない場合は、AWSの紹介のために、最初にそれを行うことを検討してくださいAWS SDK for .NET。

クロスプラットフォームでの開発

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。

- （オプション）AWSツールキットは、選択したエディタとオペレーティングシステムで使用できる場合はそのツールキットです。

  例を以下に示します。AWS Toolkit for Visual Studio Code,AWS Toolkit for JetBrains, およびAWS Toolkit for Visual Studio。

Visual Studio および .NET Core を使用した Windows

Visual Studio および .NET Core を使用した Windows での開発では以下が必須:

- 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含められます。

- （オプション）AWS Toolkit for Visual Studio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。これは、AWSVisual Studio からリソースとローカルプロファイルを使用します。ツールキットをインストールするには、」のセットアップAWS Toolkit for Visual Studio。

  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

プロジェクトをセットアップする (p. 17)

をセットアップするAWS SDK for .NETプロジェクト

に加えて環境をセットアップする (p. 15)では、作成する各プロジェクトを構成する必要があります。

アプリケーションがアクセスする必要がある重要な事項がいくつかありますAWSサービスをAWS SDK for .NET:

- 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資格情報、またはそのロールを引き受けるための資格情報
- の仕様AWSリージョン
- AWSSDK パッケージまたはアセンブリ

このセクションのトピックの一部では、これらの重要な設定方法について説明します。

このセクションおよび他のセクションの他のトピックでは、プロジェクトを構成するためのより高度な方法に関する情報を提供します。

トピック

- 新しいプロジェクトを開始する (p. 17)
- ユーザーとロールの作成 (p. 18)
- 設定AWS認証情報 (p. 20)
- を設定するAWSリージョン (p. 27)
- NuGet を使用した AWSSDK パッケージのインストール (p. 29)
- NuGet を使わずに AWSSDK アセンブリをインストールする (p. 30)
- の高度な設定AWS SDK for .NETプロジェクト (p. 31)

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するには、いくつかのテクニックを使用できます。AWSのサービス。次に、これらのテクニックの一部を示します。

- .NET 開発を初めて使用する場合は、AWSに追加するか、少なくとも新しいAWS SDK for .NETでは、完全な例をクイックスタート (p. 5)。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、AWS のさまざまなサービス用の C# プロジェクトテンプレートが含まれています。あなたの後ツールキットのインストールVisual Studio では、これらのテンプレートにアクセスして新しいプロジェクトを作成できます。

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

- Windows上でVisual Studioを使用して開発しても、AWS Toolkit for Visual Studioでは、一般的な手法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、ファイル,新規,プロジェクト。「.net core" を検索し、コンソールアプリケーション (.NET Core)またはWPFアプリケーション (.NET Core)テンプレートテンプレートを作成します。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。

プロジェクトを作成したら、次の適切なタスクを実行します。プロジェクトをセットアップするには (p. 17)。

で作業する方法の例をいくつか見ていきますAWSのサービスの使用AWSのサービス (p. 69)。

ユーザーとロールの作成

の結果として作成AWSアカウント (p. 15)には、（少なくとも）2つのユーザーアカウントがあります。

- ルートユーザーアカウントは、あなたのために作成され、すべてにフルアクセスできます。
- 管理者ユーザーアカウント。作成したユーザーアカウントは、ほぼすべて。

これらのユーザーアカウントは、どちらもAWSで.NET アプリケーションを実行するためのAWS。そのため、これらのタスクに適したユーザーアカウントとサービスロールを作成する必要があります。

作成する特定のユーザーアカウントとサービスロール、およびその使用方法は、アプリケーションの要件によって異なります。次に、最も単純なタイプのユーザーアカウントとサービスロールと、それらが使用される理由と作成方法を示します。

ユーザーアカウント

長期的な資格情報を持つユーザーアカウントを使用して、AWSサービスをアプリケーションを通じて実行します。このタイプのアクセスは、単一のユーザーがアプリケーションを使用する場合に適しています（たとえば、あなた）。このタイプのアクセスを使用する最も一般的なシナリオは開発中ですが、他のシナリオも可能です。

ユーザーを作成するプロセスは、状況によって異なりますが、本質的に次のようになります。

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. [Users] を選択して、[Add] を選択します。
3. ユーザー名を指定します。
4. []SelectAWSアクセスのタイプ[] を選択し、プログラムによるアクセス[]、[] の順に選択します。次へ: アクセス許可.

5. 選択既存のポリシーを直接アタッチします。[] を選択し、[] を選択します。適切なポリシー向けのAWSサービスを使用します。

   Warning

   DOないを選択します。AdministratorAccessポリシーは、アカウント内のほぼすべてのものに対する読み取りと書き込みのアクセス許可が有効になるためです。

6. [Next: (次へ:)] を選択します タグ[] を選択して、任意のタグを入力します。

   タグについては、」を参照してください。を使用してアクセスをコントロールするAWSリソースタグのIAM ユーザーガイド。

7. [Next: (次へ:)] を選択します 確認[]、[] の順に選択します。ユーザーの作成。
8. 新しいユーザーの資格情報を記録します。これは、クリアテキストをダウンロードすることで実行できます.csvファイルをコピーして貼り付けるか、アクセスキー IDおよびシークレットアクセスキー。

   これらは、アプリケーションに必要な認証情報です。

   Warning

   を使用する適切なセキュリティ対策これらの認証情報を安全に保管し、ローテーションしてください

IAM ユーザーに関する高レベル情報は、「」を参照してください。ID (ユーザー、グループ、ロール)のIAM ユーザーガイド。ユーザーに関する詳細情報は、そのガイドのIAM ユーザートピック

サービスロール

あなたは、設定することができますAWSアクセスするのサービスロールAWSサービスをユーザーに代わって提供します。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に適しています。たとえば、この目的のために作成した Amazon EC2 インスタンスなどです。

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、本質的に次のようになります。

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. 選択AWSのサービス] を選択し、[] を選択します。EC2(など) を選択し、[] を選択します。EC2ユースケース (例えば) 。
4. [Next: (次へ:)] を選択します アクセス許可を選択し、適切なポリシー向けのAWSサービスを使用します。

   Warning

   DOないを選択します。AdministratorAccessポリシーは、アカウント内のほぼすべてのものに対する読み取りと書き込みのアクセス許可が有効になるためです。

5. [Next: (次へ:)] を選択します タグ[] を選択して、任意のタグを入力します。

   タグについては、」を参照してください。を使用してアクセスをコントロールするAWSリソースタグのIAM ユーザーガイド。

6. [Next: (次へ:)] を選択します 確認を提供し、ロール名およびロールの説明。続いて、[Create role] を選択します。

IAM ロールの概要情報については、」を参照してください。ID (ユーザー、グループ、ロール)のIAM ユーザーガイド。ロールの詳細については、そのガイドのIAM; ロールトピック

設定AWS認証情報

あなたの後作成するAWSアカウント (p. 15)および必要なユーザーアカウントを作成するには (p. 18)、これらのユーザーアカウントの認証情報を管理できます。このガイドの多くのタスクと例を実行するには、これらの認証情報が必要になります。

認証情報の管理と使用の概要的なプロセスは次のとおりです。

1. 必要な認証情報を作成します。
   - ユーザーアカウントを作成するときに、認証情報を作成できます。(例については、「ユーザーアカウント (p. 18)」を参照してください。)
   - 既存のユーザーアカウントの認証情報を作成することもできます。「」を参照してください。IAM ユーザーのアクセスキーを管理します。。
2. 資格情報を格納します (たとえば、共有AWS認証情報ファイル (p. 21)またはSDK ストア (p. 23)).
3. プロジェクトを構成して、アプリケーションが認証情報の検索 (p. 25)。

次のトピックでは、環境内で資格情報を管理および使用する方法を決定するために使用できる情報を提供します。

トピック

認証情報に関する重要な警告とガイダンス

認証情報の警告

- 使用しないアカウントのルート認証情報を使用してAWSリソースの使用料金を見積もることができます。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。
- 使用しないアプリケーションファイルにリテラルアクセスキーを配置します。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。
- 使用しないには、プロジェクト領域に認証情報を含むファイルが含まれます。
- クレデンシャル-ストレージメカニズムのいずれか、共有されたAWS認証情報ファイルは、プレーンテキスト形式で保存されます。

認証情報を安全に管理するための追加のガイダンス

安全に管理する方法の全般的な説明AWSクレデンシャルについては、管理のベストプラクティスAWSアクセスキーのAWS全般のリファレンス。その説明に加えて、次の点を考慮してください。

- IAM ユーザーを作成し、IAM ユーザーをAWSroot ユーザー。IAM ユーザーの認証情報は、必要に応じて取り消すことができます。さらに、特定のリソースとアクションにアクセスできるポリシーを、各 IAM ユーザーに適用することもできます。

- を使用するタスク用の IAM ロールAmazon Elastic Container Service (Amazon ECS) タスクを実行します。

- を使用するIAM; ロールAmazon EC2 インスタンスで実行中のアプリケーション用の Amazon EC2 インスタンスです。

- を使用する一時的な認証情報 (p. 26)組織外部のユーザーで使用できるアプリケーションの環境変数です。

共有AWS認証情報ファイル

(必ず認証情報に関する重要な警告とガイダンス (p. 20)。)

アプリケーションの認証情報を提供する 1 つの方法は、共有AWS認証情報ファイルし、それらのプロファイルに資格情報を格納します。このファイルは、他のAWSSDK。また、使用できるAWS CLIとすると、AWS Tools for Windows PowerShellであり、AWSToolkit forVisual Studio,JetBrains, およびVSコード。

一般情報

デフォルトでは、共有されたAWS認証情報ファイルは、`.aws`ディレクトリにあり、という名前になっています。`credentials`である。つまり、`~/.aws/credentials`(Linux または macOS) または`%USERPROFILE%\.aws\credentials`(Windows)。代替ロケーションの詳細については、「」を参照してください。共有ファイルの場所のAWSSDK とツールのリファレンスガイド。また、「アプリケーション内の認証情報およびプロファイルへのアクセス (p. 217)」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形式のファイルで、特定の形式に従います。の形式の詳細については、「」を参照してください。AWS認証情報ファイルの詳細については、認証情報ファイルの形式のAWSSDK とツールのリファレンスガイド。

プロファイルは、共有されたAWS認証情報ファイルは、いくつかの方法で作成します。

- 任意のテキストエディタを使用して、共有されたAWS使用されます。
- を使用するAmazon.cruntime.CredentialManagement名前空間AWS SDK for .NETAPI (このトピックのAPI)。
- コマンドと手順を使用して、AWS Tools for PowerShellとAWSToolkit forVisual Studio,JetBrains, およびVSコード。
- を使用するAWS CLIコマンドを実行します。たとえば、`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、共有されたAWS使用されます。例の中には、前述したクレデンシャル管理方法のいずれかを使用して取得できる結果を示すものがあります。その他の例では、特定のメソッドの使用方法を示します。

デフォルトのプロファイルです。

共有AWS認証情報ファイルには、ほとんどの場合、default。このような場合は、AWS SDK for .NETは、他のプロファイルが定義されていない場合に資格情報を検索します。

-`[default]`プロファイルは通常、次のようになります。

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

プログラムでプロファイルを作成する

この例では、プロファイルを作成し、共有されたAWS認証情報ファイルをプログラムで実行します。これは、次のクラスを使用します。Amazon.cruntime.CredentialManagement名前空間: 資格情報のプロファイルオプション,クレデンシャルプロファイル, およびSharedCredentialsFile。

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

// For illustrative purposes only--do not include credentials in your code.
WriteProfile("my_new_profile", "AKIAIOSFODNN7EXAMPLE", "wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY");
...

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存在してはいけません。アプリケーションに含める場合は、コード内、ネットワーク上、またはコンピュータのメモリ内でプレーンテキストのキーが表示されないように、適切な予防措置を講じてください。

この例で作成されるプロファイルは、次のとおりです。

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

既存のプロファイルをプログラムで更新する

この例では、以前に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。これは、次のクラスを使用します。Amazon.cruntime.CredentialManagement名前空間: クレデンシャルプロファイルおよびSharedCredentialsFile。また、の使用RegionEndpointクラスのAmazon名前空間。

```
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);
    }
```

```
}
```

以下は、更新されたプロファイルです。

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

Note

また、設定できるAWS他の場所で、他の方法を使用してリージョン。詳細については、「を設定するAWSリージョン (p. 27)」を参照してください。

SDK ストアの使用 (Windows のみ)

(必ず重要な警告とガイドライン (p. 20)。)

Windows の場合、SDK ストアは、プロファイルを作成し、AWS SDK for .NETアプリケーションをデプロイします。これは、`[%USERPROFILE%\AppData\Local\AWSToolkit\RegisteredAccounts.json`。SDK ストアは、開発中に共有AWS認証情報ファイル (p. 21)。

一般情報

SDK ストアには次の利点があります。

- SDK ストアの認証情報は暗号化され、SDK ストアはユーザーのホームディレクトリに置かれます。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とAWS Toolkit for Visual Studio。

SDK ストアプロファイルは、特定ホスト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。つまり、開発用マシンの SDK Store にあるプロファイルは、他のホストや他の開発者のマシンで再利用することはできません。また、運用アプリケーションで SDK ストアプロファイルを使用できないことも意味します。

SDK ストアでプロファイルは、次の方法で管理できます。

- グラフィカルユーザーインターフェイス (GUI) は、AWS Toolkit for Visual Studio。
- を使用するAmazon.cruntime.CredentialManagement名前空間AWS SDK for .NETAPI (このトピックのAPI)。
- のコマンドを使用するAWS Tools for Windows PowerShellと指定します。たとえば、`Set-AWSCredential`および`Remove-AWSCredentialProfile`。

プロファイル管理の例

この例では、SDK ストアでプロファイルをプログラムで作成して更新する方法を示します。

プログラムでプロファイルを作成する

この例では、プロファイルを作成し、プログラムで SDK ストアに保存する方法を説明します。これは、次のクラスを使用します。Amazon.cruntime.CredentialManagement名前空間: 資格情報のプロファイルオプション,クレデンシャルプロファイル, およびNetSDKCredentialsFile。

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

```
...

// For illustrative purposes only--do not include credentials in your code.
WriteProfile("my_new_profile", "AKIAIOSFODNN7EXAMPLE", "wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY");
...

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存在してはいけません。アプリケーションに含まれている場合は、コード内、ネットワーク上、またはコンピュータのメモリ内にプレーンテキストのキーが表示されないように、適切な予防措置を講じてください。

この例で作成されるプロファイルは、次のとおりです。

```
"[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をプログラムで更新する

この例では、以前に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。これは、次のクラスを使用します。Amazon.cruntime.CredentialManagement名前空間: クレデンシャルプロファイルおよびNetSDKCredentialsFile。また、の使用RegionEndpointクラスのAmazon名前空間。

```
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);
    }
}
```

以下は、更新されたプロファイルです。

```
"[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

また、設定できるAWS他の場所で、他の方法を使用してリージョン。詳細については、「を設定するAWSリージョン (p. 27)」を参照してください。

認証情報とプロファイルの解決

-AWS SDK for .NETでは、特定の順序で認証情報が検索され、現在のアプリケーションで使用可能な最初のセットが使用されます。

認証情報の検索順序

1. 明示的に設定されている認証情報は、AWSサービスクライアントの詳細については、」アプリケーション内の認証情報およびプロファイルへのアクセス (p. 217)。

   Note

   そのトピックは特別な考慮事項 (p. 216)セクションを参照してください。これは、認証情報を指定するための好ましい方法ではないためです。
2. の値で指定された名前を持つ認証情報プロファイル。awsConfigs.awsProfilename。
3. で指定された名前を持つ認証情報プロファイル。`AWS_PROFILE`環境変数。
4. `[default]` 認証情報プロファイル。
5. セッション・スクリュー・アイテムから作成した、`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生成される。
6. BasicAWSCredentialsから作成した、`AWS_ACCESS_KEY_ID`および`AWS_SECRET_ACCESS_KEY`環境変数が両方とも空でない場合に生成される。
7. タスク用の IAM ロールAmazon ECS タスクの場合。
8. Amazon EC2 インスタンスメタデータ。

アプリケーションが Amazon EC2 インスタンスで実行されている場合 (実稼働環境など) 、IAM ロールを使用します。IAM ロールを使用してアクセスを許可する (p. 151)。それ以外の場合 (プレリリーステストなど) は、AWS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認証情報ファイル形式。

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それらを使用するには、それを使用します。-awsConfigs.awsProfilesロケーションプロパティは、AWS SDK for .NETはクレデンシャルプロファイルを検出します。

AWSプロファイルロケーション	プロファイルの解決動作
null (未設定) または空	プラットフォームがサポートしている場合は、SDK ストアを検索し、共有のAWS認証情報ファイルを既定の場所 (p. 21)。プロファイルがこれらの場所にない場合は、`~/.aws/config`(Linux または macOS) または`%USERPROFILE%\.aws\config`(Windows)。

AWSプロファイルロケーション	プロファイルの解決動作
ファイルへのパスAWS認証情報ファイル形式	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。

フェデレーションユーザーアカウントの認証情報を使用する

使用するアプリケーションAWS SDK for .NET(AWSSDK.Coreバージョン 3.1.6.0 以降) では、Active Directory フェデレーションサービス (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、AWSSecurity Assertion Markup Language (SAML) を使用してサービスします。

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のサービス。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。この方法については、『AWS Tools for Windows PowerShellドキュメント。

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。これを行うにはいくつかの方法があります。その一つは、awsConfigs.awsProfilenameプロパティを、他の認証情報プロファイルと同じ方法で作成します。

-AWS Security Token Serviceアセンブリ (AWSSDK.SecurityToken) は SAML サポートを提供し、AWS認証情報。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使用できることを確認してください。

ロールまたは一時的な認証情報の指定

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 ロールを使用することです。IAM ロールを使用してアクセスを許可する (p. 151)。

組織外部のユーザーで使用できるアプリケーションシナリオでは、組織外部のユーザーの場合、一時的な認証情報。への制限されたアクセスを提供することに加えてAWSリソースの場合、これらの認証情報は、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

お使いのソフトウェアがAWSプロキシを介して、プロキシの認証情報を指定するには、`ProxyCredentials`プロパティの`Config`クラスのクラスです。`-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。次に例を示します。AmazonCloudDirectoryConfigのAmazon.cloudDirectory名前空間、およびAmazonGameLiftConfigのamazon.GameLift名前空間。

を使用する場合Amazon S3たとえば、の場合、次のようなコードを使用できます。ここで、{my-username} と {my-password} は、NetworkCredentialオブジェクト。

```
AmazonS3Config config = new AmazonS3Config();
```

```
config.ProxyCredentials = new NetworkCredential("my-username", "my-password");
```

Note

SDK の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廃止されました。

を設定するAWSリージョン

AWSリージョンでアクセス可能AWS特定の地理的リージョンに物理的に存在するサービス。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。

それぞれの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サービス、「」を参照してくださいサービスエンドポイントとクォータのAWS全般のリファレンス。既存のリージョナルエンドポイントのリストを表示するには、を参照してください。AWSサービスエンドポイント。リージョンの詳細については、「」を参照してください。管理AWSリージョン。

作成することができます。AWSにアクセスするサービスクライアント特定のリージョン (p. 27)。また、次のリージョンでアプリケーションを構成することもできます。すべてAWSサービスクライアント (p. 28)。次に、この 2 つのケースについて説明します。

特定のリージョンでサービスクライアントを作成する

リージョンは、次のことに注意してください。AWSアプリケーションのサービスクライアント。このようにリージョンを設定すると、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、のインスタンス化方法を示します。Amazon EC2 クライアント既存のリージョン内。定義済みを使用しますRegionEndpointフィールド。

```
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、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ます。RegionEndpoint.Get by システム名。

```
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構成クラスを使用した新しいリージョン

この例では、以下の方法を示します。ServiceURLリージョンを指定するためのサービスクライアント設定クラスのプロパティ。この場合は、Amazonec2Configクラス。

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のリージョンを指定する

すべてのリージョンを指定する方法は複数あります。AWSアプリケーションが作成するサービスクライアント。このリージョンは、特定のリージョンで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。

Profiles

アプリケーションまたは SDK がロードし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 (p. 25)」を参照してください。

環境変数

に設定するAWS_REGION環境変数。

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した場合 (exportまたはsetx) の場合、すべての SDK とツールキットに影響します。AWS SDK for .NET。

AWS Configs クラス

として設定AWSConfigs.awsRegionプロパティ。

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

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。他のアカウントと認証情報AWSリージョンは、中国 (北京) リージョンでは使用されません。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地域。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、「中国 (北京) リージョン」を参照してください。

新しいに関する特別な情報AWSのサービス

新規AWS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。前述のように、新しく追加されたリージョンは、クライアントごと、またはグローバルに指定できます。

NuGet を使用した AWSSDK パッケージのインストール

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。NNuGet を使用すると、AWSSDK パッケージ、その他のいくつかの拡張機能をプロジェクトに追加します。詳細については、「」を参照してください。aws/ドットネットGitHub に移動します。

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 ではパッケージ間の依存関係が認識され、必要なすべてのパッケージが自動的にインストールされます。

> 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) が含まれる場合があります。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り、新しいプロジェクトに使用しないでください。

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
>
> プロジェクト単位で 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 アセンブリを取得し、ローカル (またはオンプレミス) に格納できます。
> これが該当し、AWSSDK アセンブリをまだ取得していない場合は、AWSSDK アセンブリを入手する (p. 216)。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 を使わずに AWSSDK アセンブリをインストールする (p. 30)。

Command プロンプトまたはターミナルからの NuGet の使用

1. に移動しますNuGet での AWSSDK パッケージプロジェクトに必要なパッケージを決定します。次に例を示します。AWSSDK.S3。
2. 次の例に示すように、そのパッケージの Web ページから .NET CLI コマンドをコピーします。

```
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

   NuGet パッケージの最新バージョンだけが必要な場合は、次の例に示すように、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。

   次の図は、AWSSDK.S3パッケージ化する。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。

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、ツール,NuGet パッケージマネージャー,パッケージマネージャーコンソール。

パッケージマネージャーコンソールで、必要な AWSSDK パッケージをインストールするには、**`Install-Package`**コマンドを実行します。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`オプションを選択して、次の例のように、希望するパッケージのバージョンを指定します。

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

パッケージマネージャーコンソールのコマンドの詳細については、「PowerShell リファレンスMicrosoft のNuGet ドキュメント。

NuGet を使わずに AWSSDK アセンブリをインストールする

このトピックでは、取得してローカル (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。詳細については、AWSSDK アセンブリを入手する (p. 216)。これはないSDK参照を処理するための推奨される方法ですが、一部の環境では必要です。

Note

SDK リファレンスを処理するための推奨される方法は、各プロジェクトに必要な NuGet パッケージだけをダウンロードしてインストールすることです。この方法については、NuGet を使用した AWSSDK パッケージのインストール (p. 29)。

AWSSDK アセンブリをインストールするには

1. 必要な AWSSDK アセンブリ用のフォルダをプロジェクトエリアに作成します。たとえば、このフォルダを`AwsAssemblies`。
2. まだの場合は、AWSSDK アセンブリを取得する (p. 216)。これにより、アセンブリが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配置されます。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にコピーします (`AwsAssemblies`フォルダ、この例では)。

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

   Cross-platform development

   1. プロジェクトの`.csproj`ファイルを編集し、`<ItemGroup>`element
   2. 左`<ItemGroup>`要素を追加するには、`<Reference>`要素を`Include`各必要なアセンブリのための属性を。

      たとえば、Amazon S3 の場合、次の行をプロジェクトの`.csproj`ファイルを開きます。

      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`ファイルを開きます。

   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,参照を追加。
   2. [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)。
   3. すべての DLL ファイルを選択し、を追加します。を選択し、OK。
   4. プロジェクトを保存します。

の高度な設定AWS SDK for .NETプロジェクト

このセクションのトピックでは、関心のある追加の構成タスクと方法について説明します。

トピック

AWSSDK.Extensions.NETCore.Setup とアイコン設定インターフェイスの使用

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

.NET Core の最大の変更点の 1 つは、`ConfigurationManager`および標準`app.config`および`web.config`.NET Framework および ASP.NET アプリケーションで使用されていたファイル。

.NET Core の設定は、設定プロバイダーによって確立されたキーと値のペアに基づいています。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を読み取ります。

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

を使いやすくするために、AWS SDK for .NET.NET Core を使用すると、AWSSDK.Extensions.NETCore.SetupNuGet パッケージです。これは多くの.NET Coreライブラリと同様に、に拡張メソッドを追加します。`IConfiguration`取得するためのインターフェイスAWSシームレスな構成。

AWSSDK.Extensions.NETCore.Setup の使用

ASP.NET コアモデル-ビューコントローラ (MVC) アプリケーションを作成するとします。これは、ASP.NET Core Web アプリケーションVisual Studioでテンプレートまたは実行して`dotnet new mvc ...`.NET Core CLI を使用します。このようなアプリケーションを作成する場合、のコンストラクタ`Startup.cs`次のような設定プロバイダからさまざまな入力ソースを読み込んで、設定を処理します。`appsettings.json`。

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

♪`Configuration`オブジェクトを取得するオブジェクトAWSオプション、まず`AWSSDK.Extensions.NETCore.Setup`NuGet パッケージです。次に、次に示すように、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 つはであることに注意してください。`appsettings.Development.json`。これは、`EnvironmentName`に設定します。開発。開発時には、ローカルテスト時にのみ読み込まれる設定を、このファイルに入れます。以下の Amazon EC2 インスタンスをデプロイする場合`EnvironmentName`に設定します。本番稼働用の場合、このファイルは無視され、AWS SDK for .NETAmazon EC2 インスタンスに設定された IAM 認証情報およびリージョンにフォールバックします。

次の設定は、に追加できる値の例を示しています。`appsettings.Development.json`プロジェクト内のファイルを提供するAWS[] 設定。

```
{
  "AWS": {
    "Profile": "local-test-profile",
    "Region": "us-west-2"
```

```
  },
  "SupportEmail": "TechSupport@example.com"
}
```

の設定にアクセスするにはCSHTMLファイル、Configurationディレクティブ。

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

<h1>Contact</h1>

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

にアクセスするには、次の手順に従います。AWSファイルで設定されたオプションはコードから で、GetAWSOptions拡張メソッドがに追加されましたIConfiguration。

これらのオプションからサービスクライアントを構築するには、CreateServiceClient を呼び出します。次の例では、Amazon S3 サービスクライアントを作成する方法を示します。(必ず、AWSSDK.S3NuGet パッケージをプロジェクトに追加します。

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

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。appsettings.Development.jsonファイル。次の例に示すように、の設定は次のとおりです。service1を含むus-west-2のリージョンと設定service2特別なエンドポイントを含むURL。

```
{
  "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。たとえば、の設定を取得するには、次のようにします。service1以下を使用します。

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

appsettings ファイルで許可される値

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。フィールド名には、表示されている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、「」を参照してください。AWS.Runtime.ClientConfigクラス。

- リージョン
- プロフィール
- ProfilesLocation
- SignatureVersion
- RegionEndpoint

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

ASP.NET Core 依存関係インジェクション

AWSSDK.Extensions.NETCore.Setup NuGet パッケージにも、ASP.NET Core の新しい依存関係インジェクションシステムが組み込まれています。-`ConfigureServices`アプリケーションのメソッド`Startup`クラスは、MVC サービスが追加されている場所です。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における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。

-`AWSSDK.Extensions.NETCore.Setup`NuGet パッケージは、新しい拡張メソッド を`IServiceCollection`追加するために使用できることAWS依存性注入へのサービス。次のコードは、を追加する方法を示しています。AWS読み込まれたオプション`IConfiguration`をクリックして、Amazon S3 と DynamoDB をサービスのリストに追加します。 (必ず、AWSSDK.S3およびAWSSDK.DynamoDBv2NuGet パッケージをプロジェクトに追加します。)

```
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。-App.configおよびWeb.configデフォルトでは、.NET Core に基づくプロジェクトには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

認証情報の設定 (p. 20)に加えて、その他のアプリケーションパラメータを数多く設定できます。

- AWSLogging (p. 35)
- AWSLogMetrics (p. 36)
- AWSRegion (p. 36)
- AWSResponseLogging (p. 37)
- AWS.DynamoDBContext.TableNamePrefix (p. 37)
- AWS.S3.UseSignatureVersion4 (p. 38)
- AWSEndpointDefinition (p. 38)
- AWSサービスによって生成されたエンドポイント (p. 39)

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。

の使用方法の詳細については、「」を参照してください。<aws>このトピックの後半で説明する要素については、を参照してください。の設定ファイルリファレンスAWS SDK for .NET (p. 40)。

AWSLogging

SDK でイベントを記録する方法を設定します。たとえば、推奨される方法は、<logging> 要素の子要素である <aws> 要素を使用することです。

```
<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。これがデフォルト値です。

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、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

デフォルトを設定します。AWS明示的にリージョンを指定していないクライアントのリージョン。.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、AWSRegionでのキー<appSettings>セクションに追加します。

```
<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";
```

の作成の詳細については、「」を参照してください。AWS特定のリージョンのクライアントについては、を参照してください。AWSリージョンの選択 (p. 27)。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。これがデフォルト値です。

`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、AWSResponseLoggingでのキー`<appSettings>`セクションに追加します。

```
<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 `TableNamePrefix` で使用されるデフォルトの `DynamoDBContext` 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で署名バージョン 4 の署名を使用するかどうかを設定します。

で Amazon S3 の署名バージョン 4 の署名を設定するには.configファイルを使用する場合、推奨される方法は、useSignatureVersion4の属性<s3>element (element) の子要素です。<aws>element

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

または、AWS.S3.UseSignatureVersion4への鍵trueの<appSettings>セクションに追加します。

```
<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、AWSEndpointDefinitionでのキー<appSettings>セクションに追加します。

```
<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。

AWSサービスによって生成されたエンドポイント

ある程度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 2 つの例としては、Amazon CloudSearch およびAWS IoT。次の例は、これらのサービスのエンドポイントを取得する方法を示しています。

Amazon CloudSearch エンドポイントの例

Amazon CloudSearch クライアントは、Amazon CloudSearch 設定サービスにアクセスするために使用されます。Amazon CloudSearch 設定サービスを使用して、検索ドメインを作成、設定、管理します。検索ドメインを作成するには、CreateDomainRequest オブジェクトを作成し、`DomainName` プロパティを指定します。リクエストオブジェクトを使用して AmazonCloudSearchClient オブジェクトを作成します。CreateDomain メソッドを呼び出します。呼び出しから返される CreateDomainResponse オブジェクトには、`DocService` および `SearchService` エンドポイントの両方を持つ `DomainStatus` プロパティが含まれます。AmazonCloudSearchDomainConfig オブジェクトを作成し、それを使用して `DocService`AmazonCloudSearchDomainClient`SearchService` クラスの および 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。-`App.config`および`Web.config`デフォルトでは、.NET Core に基づくプロジェクトには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 プロジェクトを使用できます`App.config`または`Web.config`指定するファイルAWS設定、などAWS認証情報、ログオプション、AWSサービスエンドポイント、およびAWSリージョン、およびいくつかの設定AWSAmazon DynamoDB、Amazon EC2、Amazon S3 などのサービス。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**
>
> 引き続き使用できるが`<appSettings>`内の要素`App.config`または`Web.config`指定するファイルAWS設定の場合、`<configSections>`および`<aws>`このトピックの後半で説明します。の詳細については、「」を参照してください。`<appSettings>`要素については、`<appSettings>`での要素の例の設定AWS SDK for .NETアプリケーション (p. 17)。

Note

以下を引き続き使用することはできますがAWSConfigs指定するコードファイル内のクラスプロパティAWS設定の場合、次のプロパティは廃止されており、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

一般的には、を使用する代わりに、をお勧めします。AWSConfigs指定するコードファイル内のクラスプロパティAWS設定の場合は、<configSections>および<aws>内の要素App.configまたはWeb.config指定するファイルAWSこのトピックの後半で説明します。前述のプロパティの詳細については、を参照してください。AWSConfigsでのコード例の設定AWS SDK for .NETアプリケーション (p. 17)。

トピック

- と宣言するAWS設定セクション (p. 41)
- 使用できる要素 (p. 41)
- 要素のリファレンス (p. 42)

と宣言するAWS設定セクション

指定するAWSの設定App.configまたはWeb.config内のファイル<aws>element <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

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

```
<aws
  endpointDefinition="string value"
  region="string value"
  profileName="string value"
  profilesLocation="string value">
  <logging
    logTo="None, Log4Net, SystemDiagnostics"
    logResponses="Never | OnError | Always"
    logMetrics="true | false"
    logMetricsFormat="Standard | JSON"
    logMetricsCustomFormatter="NameSpace.Class, Assembly" />
  <dynamoDB
    conversionSchema="V1 | V2">
    <dynamoDBContext
      tableNamePrefix="string value">
      <tableAliases>
        <alias
          fromTable="string value"
          toTable="string value" />
      </tableAliases>
      <map
        type="NameSpace.Class, Assembly"
        targetTable="string value">
        <property
          name="string value"
          attribute="string value"
          ignore="true | false"
          version="true | false"
          converter="NameSpace.Class, Assembly" />
      </map>
    </dynamoDBContext>
  </dynamoDB>
  <s3
    useSignatureVersion4="true | false" />
  <ec2
    useSignatureVersion4="true | false" />
  <proxy
    host="string value"
    port="1234"
    username="string value"
    password="string value" />
</aws>
```

要素のリファレンス

以下に、で使用できる要素のリストを示します。AWS[Redis settings 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

<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、の最上位要素を表します。AWS[Redis settings この要素は次の属性を含むことができます:

endpointDefinition

を定義するカスタム設定ファイルへの絶対パスです。AWS使用するリージョンおよびエンドポイント 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

保存されるプロファイル名AWSサービスコールの作成に使用される認証情報。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

他と共有される認証情報ファイルの場所への絶対パスです。AWSSDK。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

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

logging

<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

プロパティの名前です 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:

host

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" />
```

AWS SDK for .NET の機能

このセクションでは、の機能を説明します。AWS SDK for .NETで、アプリケーションを作成するときに考慮する必要がある場合があります。

以下の機能を備えています。プロジェクトをセットアップするには (p. 17)最初.

特定のソフトウェア開発については、AWSサービスの詳細については、」の使用AWSサービスをAWS SDK for .NET (p. 69)。

トピック

.NET の AWS 非同期 API

-AWS SDK for .NETを使用するタスクベースの非同期パターン (TAP)その非同期実装のために。TAP の詳細については、『』を参照してください。タスクベースの非同期パターン (TAP)docs.microsoft.com で入手できます

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

非同期メソッドのAWS SDK for .NETAPI は、`Task`クラスまたは`Task<TResult>`クラス これらのクラスの詳細については、docs.microsoft.com を参照してください。タスククラス,タスク <TResult> クラス。

これらの API メソッドがコード内で呼び出されるとき、それらは`async`次の例に示すように、キーワードを追加します。

```
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すように、`async`宣言は`Main`function. これを設定`async`スコープは、AWSサービスクライアントは非同期である必要があります。宣言できない場合`Main`何らかの理由で非同期に

なっている場合は、async以外の関数でキーワードMainAPI メソッドを呼び出すには、次の例のようにAPI メソッドを使用します。

```
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;
}
```

特別なTask<>で必要な構文Mainこのパターンを使用する場合。さらに、**Result**メンバーを呼び出してデータを取得します。

非同期呼び出しの完全な例を見ることができますAWSサービスクライアントのクイックスタート (p. 5)セクションに追加するには、シンプルなクロスプラットフォームアプリ (p. 5)およびシンプルな Windows ベースのアプリ (p. 10)) とのAWSサービス (p. 69)。

再試行とタイムアウト

-AWS SDK for .NETでは、HTTP リクエストの再試行回数とタイムアウト値の数を設定できます。AWSのサービス。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。

- どうすべきですかAWS SDK for .NETネットワーク接続の速度が低下している場合、アプリケーションが応答するか、AWSサービスは到達不能ですか？ 呼び出しがすぐに失敗するか、またはユーザーに代わって呼び出しが再試行され続けるか、どちらが適切か。
- 応答性が必要なユーザー対応アプリケーションまたはウェブサイトであるか、またはレイテンシーの増加に耐性があるバックグラウンド処理ジョブであるか。
- アプリケーションが低レイテンシーの信頼性の高いネットワークでデプロイされているか、または信頼性が低い接続でリモートの場所にデプロイされているか。

Retries

Overview

-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が再試行されます。サービスクライアントの再試行動作を指定するために使用できるサービス構成クラスには 2 つのプロパティがあります。サービス構成クラスは、これらのプロパティを抽象から継承します。Amazon.runtime.ClientConfigのクラスAWS SDK for .NETAPI リファレンス:

- RetryMode3 つのリトライモードのうちの 1 つを指定します。これは、Amazon.runtime.requestRetryMode列挙。

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retry_mode共有での設定 AWSconfig ファイル。

- `MaxErrorRetry`サービスクライアントレベルで許可された再試行回数を指定します。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max_attempts共有での設定 AWSconfig ファイル。

これらのプロパティの詳細については、要約を参照してください。Amazon.runtime.ClientConfigのクラスAWS SDK for .NETAPI リファレンス。の各値`RetryMode`デフォルトでは、特定の値に対応します。`MaxErrorRetry`である。次の表を参照。

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期

アプリケーションの起動時に、のデフォルト値`RetryMode`および`MaxErrorRetry`SDK によって構成されます。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。

- 環境内でプロパティが設定されていない場合は、のデフォルトです。`RetryMode`として構成されている。レガシーおよびのデフォルトはです。`MaxErrorRetry`は、前の表の対応する値で設定されています。
- 環境内で再試行モードが設定されている場合、その値がデフォルトとして使用されます。`RetryMode`。のデフォルトはです。`MaxErrorRetry`は、最大エラーの値が環境でも設定されていない限り、前の表の対応する値で設定されています (次を参照)。
- 最大エラーの値が環境で設定されている場合、その値がのデフォルトとして使用されます。`MaxErrorRetry`。Amazon DynamoDB はこのルールの例外です。デフォルトの DynamoDB 値です。`MaxErrorRetry`は常に前のテーブルの値です。

アプリケーションの実行中

サービスクライアントを作成したら、のデフォルト値を使用できます。`RetryMode`および`MaxErrorRetry`、前述のように、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、次のようなサービス設定オブジェクトを作成して含めます。AmazonDynamoDBConfigまたはAmazonSQSConfigサービスクライアントを作成するとき。

サービスクライアントの作成後にサービスクライアント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。

Timeouts

-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`と`ReadWriteTimeout`アブストラクトのプロパティAmazon.runtime.ClientConfigクラス。これらの値は、`Timeout`および`ReadWriteTimeout`のプロパティHttpWebRequestによって作成されるオブジェクトAWSサービスクライアントオブジェクト。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) を対象としている AWS SDK for .NET のバージョンでは、 クラスではなく HttpClient`HttpWebRequest` クラスが使用されていて、Timeout プロパティだけがサポートされています。

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`および`ReadWriteTimeout`呼び出されるメソッドがストリームをアップロードする場合、最大値に設定されます。Amazons3Client.putObjectaSync (),Amazons3Client.uploadPartAsync (),AmazonlacierClient.uploadArchiveAsync ()などといった具合です。
- のバージョンAWS SDK for .NET.NET Framework のターゲットセット`Timeout`および`ReadWriteTimeout`すべての最大値にAmazons3ClientおよびAmazonGlacierClientオブジェクト。
- のバージョンAWS SDK for .NETポータブルクラスライブラリ (PCL) と.NET Core セットをターゲットとする`Timeout`すべての最大値にAmazons3ClientおよびAmazonGlacierClientオブジェクト。

Example

次の例では、指定方法を示します。Standard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。-Amazons3Clientコンストラクタには、Amazons3Configオブジェクト。

```
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
  });
```

Paginators

ある程度AWSサービスは大量のデータを収集して格納します。このデータは、の API 呼び出しを使用して取得できます。AWS SDK for .NET。1 回の API 呼び出しで取得するデータ量が多すぎる場合は、次の方法を使用して、結果をより管理しやすい部分に分割できます。ページ分割。

ページネーションを実行できるようにするために、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は、継続トークン(通常名前付き)NextToken). これらのサービスクライアントの一部も提供しています。ページネーター。

ページネーターを使用すると、ループ、状態変数、複数の API 呼び出しなどを含む継続トークンのオーバーヘッドを回避できます。ページネータを使用すると、データを取得できます。AWS一行のコードを通じてサービスを行い、foreachループの宣言。データを取得するために複数の API 呼び出しが必要な場合は、ページネータがこれを処理します。

ページ分割ツールはどこで見つけられますか

すべてのサービスがページネーターを提供するわけではありません。サービスが特定の API のページネータを提供するかどうかを判断する 1 つの方法は、でサービスクライアントクラスの定義を調べることです。AWS SDK for .NETAPI リファレンス。

たとえば、の定義を調べると、AmazonCloudWatchlogsClientクラス、見えるPaginatorsプロパティ。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。

ページネーターは私に何を与えるのですか？

ページネーターには、完全な応答を表示できるプロパティが含まれています。また、通常、レスポンスの最も興味深い部分にアクセスできるようにする 1 つ以上のプロパティが含まれています。主な結果。

たとえば、AmazonCloudWatchLogsClient先に述べたように、Paginatorオブジェクトにはが含まれます。Responses完全な物件説明ロググループレスポンスAPI 呼び出しからのオブジェクト。このResponsesプロパティには、とりわけ、ロググループのコレクションが含まれます。

Paginator オブジェクトには、という名前のキー結果が 1 つ含まれています。LogGroups。このプロパティはレスポンスのロググループ部分のみを保持します。この重要な結果により、多くの状況でコードを減らして簡素化できます。

同期ページ分割と非同期ページ分割

ページネータは、ページネーションの同期メカニズムと非同期の両方のメカニズムを提供します。.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。非同期ページネーションは、.NET Core プロジェクトで使用できます。

非同期操作と.NET Core が推奨されるため、次の例では非同期ページネーションを示します。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、の例の後に説明します。ページ分割に関するその他の考慮事項 (p. 56)。

Example

次の例は、の使用方法を示しています。AWS SDK for .NETをクリックして、ロググループのリストを表示します。対照的に、この例ではページ分割を使用せずにこれを行う方法を示します。後で示す完全なコードを見る前に、次のスニペットを検討してください。

ページネーターなしで CloudWatch ロググループを取得する

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

```
        foreach(var logGroup in response.LogGroups)
        {
          Console.WriteLine($"{logGroup.LogGroupName}");
        }
        request.NextToken = response.NextToken;
      } while(!string.IsNullOrEmpty(request.NextToken));
```

ページネーターを使用して CloudWatch ロググループを取得する

```
      // No need to loop to get all the log groups--the SDK does it for us behind the
 scenes
      var paginatorForLogGroups =
        cwClient.Paginators.DescribeLogGroups(new DescribeLogGroupsRequest());
      await foreach(var logGroup in paginatorForLogGroups.LogGroups)
      {
        Console.WriteLine(logGroup.LogGroupName);
      }
```

これら2つのスニペットの結果はまったく同じであるため、ページネータを使用する際の利点が明確に見られます。

> **Note**
>
> 完全なコードをビルドして実行する前に、環境をセットアップする (p. 15)。「」の情報を確認します。プロジェクトをセットアップする (p. 17)。
> また、必要な場合もあります。Microsoft.BCL.AsyncInterfacesNuGet パッケージは、非同期ページネータが`IAsyncEnumerable`インターフェイスからリクエスト.

完全なコード

このセクションでは、関連するリファレンスとこの例の完全なコードを示します。

SDK リファレンス

NuGet パッケージ:

- awssdk.CloudWatch

プログラミング要素:

- 名前空間Amazon.CloudWatch

  クラスAmazonCloudWatchlogsClient
- 名前空間Amazon.cloudWatchlogs.Model

  クラス説明ロググループリクエスト

  クラス説明ロググループレスポンス

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

ページ分割に関するその他の考慮事項

- ページネータを複数回使うことはできない

  特定の結果が必要な場合AWSコード内の複数の場所にあるページネーターの場合、ページネーターオブジェクトを複数回使用してはいけません。代わりに、必要な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、前のサンプルコードに示されています。`DisplayLogGroupsWithPaginators`方法。

- 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。

  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、上記のコードをコピーします。次に、単に削除してください。`await`2 つのキーワードを使用します。`foreach`次の例に示すように、ページ分割ツールを呼び出します。

```
/*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て、非同期ページネーションで見た結果と同じ結果を確認します。

SDK メトリクスの有効化

AWS SDK Metrics for Enterprise Support (SDK Metrics) により、エンタープライズのお客様はAWSで共有されているホストおよびクライアント上の SDKAWSEnterprise Support SDK メトリクスは、への接続で発生する問題の検出と診断の迅速化に役立つ情報を提供します。AWSのサービスAWSEnterprise Support

テレメトリは、各ホストで収集されると、UDP 経由で 127.0.0.1 (localhost) に中継され、そこで Amazon CloudWatch エージェントによってデータが集約されて SDK メトリクスに送信されます。したがって、メトリクスを受信するには、CloudWatch エージェントをインスタンスに追加する必要があります。

SDK メトリクスをセットアップする次のステップは、Amazon Linux を実行している Amazon EC2 インスタンスに関連しています。Amazon EC2 インスタンスでは、AWS SDK for .NET。SDK メトリクスは、本番稼働用環境でも使用できます。これは、AWS SDK for .NET。

SDK メトリクスを利用するには、最新バージョンの CloudWatch エージェントを実行します。「」でその方法を説明しますSDK メトリックス用の CloudWatch エージェントの設定」を参照してください。

SDK メトリクスをセットアップするにはAWS SDK for .NETを使用するには、以下の手順に従います。

1. AWS SDK for .NET クライアントで、AWS のサービスを使用するためのアプリケーションを作成します。
2. Amazon EC2 インスタンスまたはローカル環境でプロジェクトをホストします。
3. 最新バージョンの AWS SDK for .NET をインストールして使用します。
4. EC2 インスタンスまたはローカル環境に CloudWatch エージェントをインストールして設定します。
5. メトリクスを収集して送信することを SDK メトリクスに許可します。
6. の SDK メトリクスを有効にするAWS SDK for .NET (p. 57)。

詳細については、以下を参照してください。

- CloudWatch エージェントを更新する (p. 58)
- SDK メトリクスを無効にする (p. 58)

の SDK メトリクスを有効にするAWS SDK for .NET

デフォルトでは、SDK メトリクスはオフになっており、ポートは 31000 に設定されています。デフォルトのパラメータは以下のとおりです。

```
//default values
 [
     'enabled' => false,
     'port' => 31000,
 ]
```

SDK メトリクスを有効にするのは、認証情報を設定することとは無関係でAWSサービス。

SDK メトリクスを有効にするには、環境変数を設定するか、AWS共有 Config ファイル

オプション 1: 環境変数の設定

もしAWS_CSM_ENABLEDが設定されていない場合、SDK は最初にAWS_PROFILEを使用して、SDK メトリックスが有効になっているかどうかを判断します。デフォルトでは、false に設定されています。

SDK メトリクスを有効にするには、環境変数に次のコードを追加します。

```
export AWS_CSM_ENABLED=true
```

その他の構成設定 (p. 58)が使用可能になります。

> Note
>
> SDK メトリクスを有効にしても、認証情報がAWSサービス。

オプション 2:AWS共有 Config ファイル

環境変数で SDK メトリクスを設定が見つからない場合、SDK はデフォルトのAWSプロファイルフィールドに入力します。AWS_DEFAULT_PROFILE がデフォルト以外の値に設定されている場合は、そのプロ

ファイルを更新します。SDK メトリクスを有効にするには、`~/.aws/config` にある共有 config ファイルに `csm_enabled` を追加します。

```
[default]
csm_enabled = true

[profile aws_csm]
csm_enabled = true
```

その他の構成設定 (p. 58)が使用可能になります。

> **Note**
>
> SDK メトリクスを有効にするのは、認証情報を設定することとは無関係で、AWSサービス。認証には別のプロファイルを使用できます。

CloudWatch エージェントを更新する

ポートを変更するには、値を設定して、任意のポートを再起動する必要があります。AWS現在アクティブなジョブ。

オプション 1: 環境変数の設定

ほとんどのサービスではデフォルトポートを使用します。ただし、サービスで一意のポート ID が必要な場合は、AWS_CSM_PORT=[port_number] をホストの環境変数に追加します。

```
export AWS_CSM_ENABLED=true
export AWS_CSM_PORT=1234
```

オプション 2:AWS共有 Config ファイル

ほとんどのサービスではデフォルトポートを使用します。ただし、サービスで一意のポート ID が必要な場合は、csm_port = [port_number] を ~/.aws/config に追加します。

```
[default]
csm_enabled = false
csm_port = 1234

[profile aws_csm]
csm_enabled = false
csm_port = 1234
```

SDK メトリクスを再起動

ジョブを再起動するには、以下のコマンドを実行します。

```
amazon-cloudwatch-agent-ctl -a stop;
amazon-cloudwatch-agent-ctl -a start;
```

SDK メトリクスを無効にする

SDK メトリックスをオフにするには、csm_enabled ~ false環境変数で、またはAWS共有 Config ファイル`~/.aws/config`。次に、CloudWatch エージェントを再起動して変更を反映します。

環境変数

```
export AWS_CSM_ENABLED=false
```

AWS 共有 Config ファイル

を削除します。csm_enabledのプロファイルからAWS共有 Config ファイル~/.aws/config。

> **Note**
>
> 環境変数は、AWS共有 Config ファイル 環境変数で SDK メトリクスを有効にすると、SDK メトリクスは有効なままになります。

```
[default]
csm_enabled = false

[profile aws_csm]
csm_enabled = false
```

SDK メトリクスを無効にするには、次のコマンドを使用して CloudWatch エージェントを停止します。

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a stop &&
echo "Done"
```

他の CloudWatch 機能を使用している場合は、次のコマンドを使用して CloudWatch を再起動します。

```
amazon-cloudwatch-agent-ctl -a start;
```

SDK メトリクスを再起動

ジョブを再起動するには、以下のコマンドを実行します。

```
amazon-cloudwatch-agent-ctl -a stop;
amazon-cloudwatch-agent-ctl -a start;
```

SDK メトリクスの定義

SDK メトリクスの以下の説明を参考にして結果を解釈できます。通常、これらのメトリクスは定期的なビジネスレビューでテクニカルアカウントマネージャーによるレビューに使用できます。AWSSupport のリソースおよびテクニカルアカウントマネージャーは、問題が発生したときに問題が発生したときにSDK メトリクスのデータにアクセスして解決に役立てることができますが、紛らわしいデータや予期しないデータを検出した場合は、アプリケーションのパフォーマンスに悪影響がなければ、スケジュールされたビジネスで該当するデータを確認することをお勧めします。レビュー.

メトリクス	CallCount
定義	コードからへの成功または失敗した API コールの総数AWSのサービス
使用方法	エラーやスロットリングなどの他のメトリクスと相関させるためのベースラインとして使用します。

メトリクス	ClientErrorCount
定義	クライアントエラー (4xx HTTP 応答コード) で失敗した API コールの数。例: スロットリング、アクセス拒否、S3 バケットが存在しない、無効なパラメータ値など。
使用方法	スロットリングに関連する特定の問題 (引き上げを要する制限のせいでスロットリングが発生した場合など) を除いて、このメトリクスは、アプリケーションに修正を要する問題があることを示している可能性があります。

メトリクス	ConnectionErrorCount
定義	サービスへの接続エラーにより失敗した API コールの数。これらは、お客様のアプリケーションとアプリケーションとのネットワーク問題AWSサービス (ロードバランサー、DNS の障害、トランジットプロバイダーなど)。場合によっては、AWS の問題が原因で、このエラーが発生する場合もあります。
使用方法	このメトリクスを使用して、問題の原因がアプリケーションにあるか、インフラストラクチャやネットワークにあるかを判断します。ConnectionErrorCount の値が大きい場合、原因として API コールのタイムアウト値が短いことも考えられます。

メトリクス	ThrottleCount
定義	AWS のサービスのスロットリングに伴って失敗した API コールの数。
使用方法	このメトリクスを使用して、アプリケーションがスロットリング制限に達したかどうかを評価したり、再試行やアプリケーションのレイテンシーの原因を判断したりします。コールをまとめて処理するのではなく、期間全体に分散させることを検討します。

メトリクス	ServerErrorCount
定義	からのサーバーエラー (5xx HTTP 応答コード) により失敗した API コールの数AWSサービス。これらは通常、AWS サービスによって発生します。
使用方法	SDK の再試行やレイテンシーの原因を判断します。このメトリックは、AWSサービスは、いくつかのように、障害にあるAWSチームは、レイテンシーを HTTP 503 応答として分類します。

メトリクス	EndToEndLatency
定義	アプリケーションがコールを行う合計時間AWSSDK (再試行を含む) 。つまり、数回の試行後に成功したかどうかや、取得不能エラーですぐに失敗したかどうかは関係ありません。
使用方法	方法を決定するAWSAPI コールは、アプリケーションのレイテンシー全体に左右されます。予期された以上にレイテンシーが高い場合、問題の原因はネットワーク、ファイアウォール、その他の設定にあるか、SDK の再試行の結果として発生するレイテンシーにある可能性があります。

その他のツール

次に、.NET アプリケーションの開発、配備、および保守作業を容易にする追加ツールを示します。

AWS.NET 配置ツール

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。

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

詳細については、「」を参照してください。デプロイツール (p. 190)がとともに使用する追加ツールは AWS SDK for .NET (p. 190)。

プロジェクトのマイグレーション AWS SDK for .NET

このセクションでは、適用可能な移行タスクに関する情報と、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

の新機能AWS SDK for .NET

製品ページはhttps://aws.amazon.com/sdk-for-net/に関連する新しい開発の概要情報についてはAWS SDK for .NET。

の新機能は次のとおりです。AWS SDK for .NET。

2021 年 3 月 17 日: 開発者プレビューAWS.NET 配置ツール

> これはプレビューリリースのサービス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.NET CLI の配置ツールである.NET 配置ツールは、開発者プレビューで使用できます。デプロイメントツールを使用して、.NET CLI からクラウドネイティブの .NET アプリケーションをデプロイできます。

2020-08-24: バージョン 3.5 の SDK SDK 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が進めます。詳細については、「バージョン 3.5 への移行 (p. 65)」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を追加しました。これにより、API結果のページネーションがより便利になりました。詳細については、Paginators (p. 52)を参照してください。

でサポートされているプラットフォームAWS SDK for .NET

AWS SDK for .NET 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 用に記述されたアプリケーションをサポートしています。AWSサービスクライアントは、NET Core の非同期の呼び出しパターンのみをサポートしています。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

フレームワーク以外のバリエーションAWS SDK for .NET準拠.NET Standard 2.0。

.NET Framework 4.5

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、非同期で待機するで導入されたキーワードC# 5.0。

.NET Framework 3.5

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。このような環境で FIPS 準拠の実装を代用する方法の詳細については、Microsoft のブログの「CryptoConfig」と、CLR Security チームの Security.Cryptography.dll 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、Universal Windows Platform (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。フレームワークの使用の詳細については、Mobile SDK for .NET and Xamarin詳細については、をご覧ください。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 (p. 219)。

詳細情報

バージョン 3.5 のに移行します。AWS SDK for .NET (p. 65)

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次のものが追加されています。WinRTとWindows Phoneかな

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。アセンブリだけを追加できますAWSサービスを使用します。

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`パッケージに含まれ、DynamoDB 項目内の参照であるオブジェクトを Amazon S3 に格納するために使用されています。これは便利な機能ですが、コンパイル依存関係を`Amazon.S3`パッケージを DynamoDB で使用しています。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。

AWSConfig 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。バージョン 3 のAWS SDK for .NETでは、DynamoDB の設定をAmazon.AWSConfigsクラスをAmazon.AWSConfigsDynamoDBクラス。

バージョン 3.5 のに移行します。AWS SDK for .NET

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、CPPをビルドするために、コードストリッピングを無効にするには、link.xmlで説明したファイル参照するAWS SDK for .NETユニティ、ザマリン、またはUWPからの標準2.0。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。より良い移行を提供するために、すべてのlegacyUnity コードでは、aws/aws-sdk-unity-netGitHub リポジトリです。およびの使用に関連する機能が不足している場合は、AWSUnity では、機能のリクエストをhttps://github.com/aws/dotnet/issues。

また、「Unity のサポートに関する特別な考慮事項 (p. 219)」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、以下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 同期マネージャーと 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、AWS SDK for .NET。AWS AppSyncAmazon Cognito Sync の代わりに、推奨されます。Amazon Pinpoint は、Amazon Mobile Analytics クスの代替手段として優先されます。

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、AWS AppSyncおよび Amazon Pinpoint では、GitHub 問題 (GitHub 問題) のいずれかまたは両方で報告できます。https://github.com/aws/dotnet/issues/20およびhttps://github.com/aws/dotnet/issues/19。Amazon Cognito 同期マネージャーと Amazon Mobile Analytics マネージャーのライブラリは、以下の GitHub リポジトリから取得することもできます。aws/アマゾン-コグニート-sync-manager-netおよびaws/aws-モバイル分析-マネージャネット。

同期コードの移行

バージョン 3.5 の AWS SDK for .NET は、AWS のサービスへの非同期呼び出しのみをサポートします。バージョン 3.5 を使用して実行する同期コードは、非同期で実行するように変更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 のに移行します。AWS SDK for .NET

バージョン 3.7 の時点で、AWS SDK for .NETは .NET Standard 1.3 をサポートしていません。

.NET Standard 1.3 からの移行については、」.NET Standard 1.3 からの移行 (p. 67)。

.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。この発表に続いて、AWSでの .NET Standard 1.3 のサポートが終了しました。AWS SDK for .NET2020 年 12 月 31 日

AWSは、サービスの更新とセキュリティ修正を提供し続けたAWS SDK for .NET2020 年 10 月 1 日まで、.NET Standard 1.3 この日以降、.NET Standard 1.3 ターゲットはメンテナンスモードになり、新しい更新はリリースされませんでした。AWSは、重大なバグ修正とセキュリティパッチのみを適用しました。

2020 年 12 月 31 日に、.NET Standard 1.3 のサポートがAWS SDK for .NET人生のその終わりに来た。この日以降、バグ修正やセキュリティパッチは適用されません。そのターゲットで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。

の使用AWSサービスをAWS SDK for .NET

次のセクションでは、使用する方法の例を示します。AWS SDK for .NETの使用するAWSのサービス。

を初めて使用する場合は、AWS SDK for .NET使用する場合は、クイックスタート (p. 5)トピックを最初に参照してください。SDK の概要を説明しています。

より多くのコード例については、AWSコード例リポジトリとawslabs リポジトリGitHub で。

開始する前に、必ず環境をセットアップする (p. 15)。また、「」の情報を確認します。プロジェクトをセットアップする (p. 17)およびSDK の機能 (p. 49)。

トピック

- へのアクセスAWS CloudFormationとAWS SDK for .NET (p. 69)
- Amazon DynamoDB NoSQL データベースの使用 (p. 71)
- Amazon EC2 での作業 (p. 91)
- へのアクセスAWS Identity and Access Managementと (IAM)AWS SDK for .NET (p. 133)
- Amazon Simple Storage Service インターネットストレージの使用 (p. 156)
- Amazon Simple Notification サービスを使用したクラウドからの通知の送信 (p. 163)
- Amazon SQS を使用したメッセージング (p. 166)
- プログラミングAWS OpsWorksするには、スタックとアプリケーションを処理する (p. 188)
- その他の SupportAWSのサービスと設定 (p. 189)

へのアクセスAWS CloudFormationとAWS SDK for .NET

-AWS SDK for .NETが をサポート AWS CloudFormation を作成し規定している。AWS計画どおりに、再現性を備えたインフラストラクチャデプロイです。

APIs

-AWS SDK for .NETの API を提供します。AWS CloudFormationクライアント。API を使用して、AWS CloudFormationテンプレートやスタックなどの機能。このセクションでは、これらの API を操作するときに従うことができるパターンを示す少数の例を紹介します。API の完全なセットを表示するには、を参照してください。AWS SDK for .NETAPI リファレンス(そして「Amazon.CloudFormation」までスクロールします)。

- AWS CloudFormation API は、awssdk.CloudFormationパッケージ化する。

Prerequisites

始める前に、使用する環境をセットアップする (p. 15)。また、「」の情報を確認します。プロジェクトをセットアップする (p. 17)およびSDK の機能 (p. 49)。

Topics

トピック

出品AWS使用するリソースAWS CloudFormation

この例では、AWS SDK for .NETリソースを一覧表示するにはAWS CloudFormationスタック。この例では、低レベルの API を使用します。アプリケーションは引数を取りませんが、ユーザーの資格情報にアクセスできるすべてのスタックの情報を収集し、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Amazon CloudFormation クライアント
- 名前空間Amazon.cloudFormation.Model

  クラスDecribeStackResourcesリクエストを記述

  クラスタック・リソース・レスポンスの説明

  クラス説明サックスレスポンス

  クラススタック

  クラスStackResource

  クラスのタグ付け

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

namespace CFNListResources
{
  class Program
  {
    static async Task Main(string[] args)
    {
      // Create the CloudFormation client
      var cfnClient = new AmazonCloudFormationClient();

      // List the resources for each stack
      await ListResources(cfnClient, await cfnClient.DescribeStacksAsync());
    }

    //
    // Method to list stack resources and other information
    private static async Task ListResources(
```

```
      IAmazonCloudFormation cfnClient, DescribeStacksResponse responseDescribeStacks)
    {
      Console.WriteLine("Getting CloudFormation stack information...");

      foreach (Stack stack in responseDescribeStacks.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 cfnClient.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----------------------------------------------------");
    }
  }
}
```

Amazon DynamoDB NoSQL データベースの使用

> **Note**
>
> このトピックの情報は、.NET Framework およびAWS SDK for .NETバージョン 3.3 以前です。

-AWS SDK for .NETは Amazon DynamoDB をサポートします。Amazon DynamoNoSQL は、AWS。SDK には、DynamoDB との通信用に 3 つのプログラムモデルが含まれます。低レベルのモデル、documentモデル、およびオブジェクトの永続性モデル。

以下では、これらのモデルとその API を紹介し、どのような場合にどのような方法で使用するかを例で示します。また、AWS SDK for .NET。

トピック

低レベルモデル

低レベルプログラミングモデルは、DynamoDB サービスに対する直接呼び出しをラップしています。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。たとえば、.NET データ型を DynamoDB の同等のデータ型に変換する必要があります。ただし、このモデルを使用するとほとんどの機能にアクセスできます。

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

テーブルの作成

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

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();

Console.WriteLine("Getting list of tables");
List<string> currentTables = client.ListTables().TableNames;
Console.WriteLine("Number of tables: " + currentTables.Count);
if (!currentTables.Contains("AnimalsInventory"))
{
    var request = new CreateTableRequest
    {
        TableName = "AnimalsInventory",
        AttributeDefinitions = new List<AttributeDefinition>
      {
        new AttributeDefinition
        {
          AttributeName = "Id",
          // "S" = string, "N" = number, and so on.
          AttributeType = "N"
        },
        new AttributeDefinition
        {
          AttributeName = "Type",
          AttributeType = "S"
        }
      },
        KeySchema = new List<KeySchemaElement>
      {
        new KeySchemaElement
        {
          AttributeName = "Id",
          // "HASH" = hash key, "RANGE" = range key.
          KeyType = "HASH"
        },
        new KeySchemaElement
        {
          AttributeName = "Type",
          KeyType = "RANGE"
        },
      },
        ProvisionedThroughput = new ProvisionedThroughput
        {
```

```
            ReadCapacityUnits = 10,
            WriteCapacityUnits = 5
        },
    };

    var response = client.CreateTable(request);

    Console.WriteLine("Table created with request ID: " +
      response.ResponseMetadata.RequestId);
}
```

テーブルを変更する準備が整っていることの確認

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

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();
var status = "";

do
{
  // Wait 5 seconds before checking (again).
  System.Threading.Thread.Sleep(TimeSpan.FromSeconds(5));

  try
  {
    var response = client.DescribeTable(new DescribeTableRequest
    {
      TableName = "AnimalsInventory"
    });

    Console.WriteLine("Table = {0}, Status = {1}",
      response.Table.TableName,
      response.Table.TableStatus);

    status = response.Table.TableStatus;
  }
  catch (ResourceNotFoundException)
  {
    // DescribeTable is eventually consistent. So you might
    //   get resource not found.
  }

} while (status != TableStatus.ACTIVE);
```

テーブルへの項目の挿入

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

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;
```

```
var client = new AmazonDynamoDBClient();

var request1 = new PutItemRequest
{
  TableName = "AnimalsInventory",
  Item = new Dictionary<string, AttributeValue>
  {
    { "Id", new AttributeValue { N = "1" }},
    { "Type", new AttributeValue { S = "Dog" }},
    { "Name", new AttributeValue { S = "Fido" }}
  }
};

var request2 = new PutItemRequest
{
  TableName = "AnimalsInventory",
  Item = new Dictionary<string, AttributeValue>
  {
    { "Id", new AttributeValue { N = "2" }},
    { "Type", new AttributeValue { S = "Cat" }},
    { "Name", new AttributeValue { S = "Patches" }}
  }
};

client.PutItem(request1);
client.PutItem(request2);
```

ドキュメントモデル

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

低レベルプログラミングと比べると、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。たとえば、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。

以下の例では、ドキュメントモデルを使用して DynamoDB のテーブルの項目を挿入および取得する方法を示します。

テーブルへの項目の挿入

次の例では、`PutItem` クラスの `Table` メソッドを使用してテーブルに項目を挿入しています。`PutItem` 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。アイテムを挿入するテーブルを決定するには、`LoadTable`のメソッド`Table`クラスのインスタンスを指定する`AmazonDynamoDBClient`クラスと DynamoDB 内でターゲットテーブル名です。

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.DocumentModel;

var client = new AmazonDynamoDBClient();
var table = Table.LoadTable(client, "AnimalsInventory");
var item = new Document();
```

```
item["Id"] = 3;
item["Type"] = "Horse";
item["Name"] = "Shadow";

table.PutItem(item);
```

テーブルからの項目の取得

次の例では、GetItem クラスの Table メソッドを使用して項目を取得しています。取得する項目を特定するために、GetItem メソッドでは対象項目のハッシュおよび範囲プライマリキーを使用しています。アイテムを取得するテーブルを決定するには、LoadTableのメソッドTableクラスのインスタンスを使用します。AmazonDynamoDBClientクラスと DynamoDB 内でターゲットテーブル名です。

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.DocumentModel;

var client = new AmazonDynamoDBClient();
var table = Table.LoadTable(client, "AnimalsInventory");
var item = table.GetItem(3, "Horse");

Console.WriteLine("Id = " + item["Id"]);
Console.WriteLine("Type = " + item["Type"]);
Console.WriteLine("Name = " + item["Name"]);
```

前の例では、Id、Type、Name の属性値を WriteLine メソッドの文字列に暗黙的に変換しています。AsType クラスのさまざまな DynamoDBEntry メソッドを使用すると、明示的に変換できます。たとえば、Id データ型の Primitive の属性値を、AsInt メソッドを使用して整数に明示的に変換できます。

```
int id = item["Id"].AsInt();
```

または、(int) を使用することで簡単に明示的なキャストを実行できます。

```
int id = (int)item["Id"];
```

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、DynamoDB での .NET オブジェクトの保存、ロード、クエリを特に目的にして作られています。このモデルには、Amazon.DynamoDBv2.DataModel 名前空間からアクセスします。

3 つのモデルのうち、オブジェクト永続性モデルは、DynamoDB データの保存、ロード、クエリに関しては常にコーディングが最も簡単です。たとえば、DynamoDB 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 の .NET オブジェクトを保存、ロード、クエリするオペレーションだけです。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。

以下の例では、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挿入する方法、.NET オブジェクトを使用して DynamoDB のテーブルから項目を取得する方法を示します。

テーブルで項目を表す .NET クラスの定義

以下の例では、DynamoDBTable 属性がテーブル名を指定しているのに対し、DynamoDBHashKey および DynamoDBRangeKey 属性はテーブルのハッシュおよび範囲プライマリキーをモデル化しています。

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

[DynamoDBTable("AnimalsInventory")]
class Item
{
  [DynamoDBHashKey]
  public int Id { get; set; }
  [DynamoDBRangeKey]
  public string Type { get; set; }
  public string Name { get; set; }
}
```

.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 Save クラスの DynamoDBContext 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 (DynamoDBContext クラスのインスタンスは、AmazonDynamoDBClient クラスのインスタンスで初期化されます)。

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

var client = new AmazonDynamoDBClient();
var context = new DynamoDBContext(client);
var item = new Item
{
  Id = 4,
  Type = "Fish",
  Name = "Goldie"
};

context.Save(item);
```

.NET オブジェクトのインスタンスの使用によるテーブルからの項目の取得

上の例では、項目は Load クラスの DynamoDBContext メソッドで取得されます。このメソッドは、取得する項目のハッシュおよび範囲プライマリキーを表す .NET クラスの部分的に初期化されたインスタンスを受け取ります (前に示したように、DynamoDBContext クラスのインスタンスは AmazonDynamoDBClient クラスのインスタンスで初期化されます)。

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

var client = new AmazonDynamoDBClient();
var context = new DynamoDBContext(client);
var item = context.Load<Item>(4, "Fish");

Console.WriteLine("Id = {0}", item.Id);
Console.WriteLine("Type = {0}", item.Type);
Console.WriteLine("Name = {0}", item.Name);
```

詳細

の使用AWS SDK for .NETDynamoDB をプログラミングする情報と例**

- DynamoDB API

- DynamoDB Series Kickoff
- DynamoDB Series - Document Model
- DynamoDB Series - Conversion Schemas
- DynamoDB Series - Object Persistence Model
- DynamoDB Series - Expressions
- Amazon DynamoDB とでの式の使用AWS SDK for .NET (p. 77)
- Amazon DynamoDB での JSON のSupportAWS SDK for .NET (p. 87)
- Amazon DynamoDB を使用した ASP.NET セッション状態の管理 (p. 88)

低レベルモデル情報と例

- を使用したテーブルの操作AWS SDK for .NET低レベル API
- を使用した項目の操作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AWS SDK for .NET低レベル API

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 およびAWS SDK for .NETバージョン 3.3 以前です。

次のコード例では、AWS SDK for .NETを使用して DynamoDB をエクスプレッションでプログラミングします。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 (p. 78)
- 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 (p. 80)
- 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 (p. 81)
- 式および他の項目属性を使って複数の項目を取得する (p. 82)
- 項目の出力 (p. 83)
- 式を使用して項目を作成または置換する (p. 84)

- 式を使用して項目を更新する (p. 85)
- 式を使用して項目を削除する (p. 86)
- 詳細 (p. 86)

サンプルデータ

このトピックのコード例では、という名前の DynamoDB テーブルの次の 2 つの項目例を使用します。ProductCatalog。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、で提供されている例に基づきます。導入事例: ProductCatalog 項目。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 (p. 83)」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 (p. 83)」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 (p. 83)」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 (p. 80)」、「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 (p. 81)」、「式および他の項目属性を使って複数の項目を取得する (p. 82)」の各方法に関する前述の例で使用されています。

```
// using Amazon.DynamoDBv2.Model;

// Writes out an item's attribute keys and values.
public static void PrintItem(Dictionary<string, AttributeValue> attrs)
{
  foreach (KeyValuePair<string, AttributeValue> kvp in attrs)
  {
    Console.Write(kvp.Key + " = ");
    PrintValue(kvp.Value);
  }
}

// Writes out just an attribute's value.
public static void PrintValue(AttributeValue value)
{
  // Binary attribute value.
  if (value.B != null)
  {
    Console.Write("Binary data");
  }
  // Binary set attribute value.
  else if (value.BS.Count > 0)
  {
    foreach (var bValue in value.BS)
    {
      Console.Write("\n  Binary data");
    }
  }
  // List attribute value.
  else if (value.L.Count > 0)
  {
    foreach (AttributeValue attr in value.L)
    {
      PrintValue(attr);
    }
  }
  // Map attribute value.
  else if (value.M.Count > 0)
  {
    Console.Write("\n");
    PrintItem(value.M);
  }
  // Number attribute value.
  else if (value.N != null)
  {
    Console.Write(value.N);
  }
  // Number set attribute value.
  else if (value.NS.Count > 0)
  {
    Console.Write("{0}", string.Join("\n", value.NS.ToArray()));
  }
  // Null attribute value.
  else if (value.NULL)
  {
    Console.Write("Null");
  }
```

```
  // String attribute value.
  else if (value.S != null)
  {
    Console.Write(value.S);
  }
  // String set attribute value.
  else if (value.SS.Count > 0)
  {
    Console.Write("{0}", string.Join("\n", value.SS.ToArray()));
  }
  // Otherwise, boolean value.
  else
  {
    Console.Write(value.BOOL);
  }

  Console.Write("\n");
}
```

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

式を使用して項目を作成または置換する

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

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();
var request = new PutItemRequest
{
  TableName = "ProductCatalog",
  ExpressionAttributeNames = new Dictionary<string, string>
  {
    { "#title", "Title" }
  },
  ExpressionAttributeValues = new Dictionary<string, AttributeValue>
  {
    { ":product", new AttributeValue { S = "18-Bicycle 301" } }
  },
  ConditionExpression = "#title = :product",
  // CreateItemData() is a custom function.
  Item = CreateItemData()
};
client.PutItem(request);
```

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

```
// using Amazon.DynamoDBv2.Model;
```

```
// Provides a sample item that can be added to a table.
public static Dictionary<string, AttributeValue> CreateItemData()
{
  var itemData = new Dictionary<string, AttributeValue>
  {
    { "Id", new AttributeValue { N = "301" } },
    { "Title", new AttributeValue { S = "18\" Girl's Bike" } },
    { "BicycleType", new AttributeValue { S = "Road" } },
    { "Brand" , new AttributeValue { S = "Brand-Company C" } },
    { "Color", new AttributeValue { SS = new List<string>{ "Blue", "Silver" } } },
    { "Description", new AttributeValue { S = "301 description" } },
    { "Gender", new AttributeValue { S = "F" } },
    { "InStock", new AttributeValue { BOOL = true } },
    { "Pictures", new AttributeValue { L = new List<AttributeValue>{
      { new AttributeValue { M = new Dictionary<string,AttributeValue>{
        { "FrontView", new AttributeValue { S = "http://example/
products/301_front.jpg" } } } } },
      { new AttributeValue { M = new Dictionary<string,AttributeValue>{
        { "RearView", new AttributeValue {S = "http://example/
products/301_rear.jpg" } } } } },
      { new AttributeValue { M = new Dictionary<string,AttributeValue>{
        { "SideView", new AttributeValue { S = "http://example/
products/301_left_side.jpg" } } } } }
    } } },
    { "Price", new AttributeValue { N = "185" } },
    { "ProductCategory", new AttributeValue { S = "Bike" } },
    { "ProductReviews", new AttributeValue { M = new Dictionary<string,AttributeValue>{
      { "FiveStar", new AttributeValue { SS = new List<string>{
        "My daughter really enjoyed this bike!" } } },
      { "OneStar", new AttributeValue { SS = new List<string>{
        "Fun to ride.",
        "This bike was okay, but I would have preferred it in my color." } } }
    } } },
    { "QuantityOnHand", new AttributeValue { N = "3" } },
    { "RelatedItems", new AttributeValue { NS = new List<string>{ "979", "822", "801" } } }
  };

  return itemData;
}
```

前の例では、サンプルデータを含む項目の例が呼び出し元に返されます。一連の属性とそれに対応する値は、次のようなデータ型を使用して構築されます。BOOL,L,M,N,NS,S, およびSSにあるものに対応する。JSON データ形式。

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

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
- を使用した項目の操作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AWS SDK for .NET低レベル API

Amazon DynamoDB での JSON のSupportAWS SDK for .NET

> Note
>
> このトピックの情報は、.NET Framework およびAWS SDK for .NETバージョン 3.3 以前です。

-AWS SDK for .NETは、Amazon DynamoDB での作業時に JSON データをサポートします。これにより、簡単に DynamoDB テーブルから JSON 形式のデータを取得したり、DynamoDB テーブルに JSON ドキュメントを挿入したりできます。

トピック

JSON 形式のデータを DynamoDB テーブルから取得する

次の例では、DynamoDB テーブルから JSON 形式のデータを取得する方法を示します。

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.DocumentModel;

var client = new AmazonDynamoDBClient();
var table = Table.LoadTable(client, "AnimalsInventory");
var item = table.GetItem(3, "Horse");

var jsonText = item.ToJson();
Console.Write(jsonText);

// Output:
//   {"Name":"Shadow","Type":"Horse","Id":3}

var jsonPrettyText = item.ToJsonPretty();
Console.WriteLine(jsonPrettyText);

// Output:
//   {
//     "Name" : "Shadow",
//     "Type" : "Horse",
//     "Id"   : 3
//   }
```

この例では、`Document` クラスの `ToJson` メソッドを使用してテーブルの項目を JSON 形式の文字列に変換しています。項目を取得するには、`Table` クラスの `GetItem` メソッドを使用します。取得する項目を特定するには、この例では、対象項目のハッシュおよび範囲プライマリキーを `GetItem` メソッドで使用しています。アイテムを取得するテーブルを決定するには、`LoadTable`のメソッド`Table`クラスのインスタンスを使用します。`AmazonDynamoDBClient`クラスと DynamoDB 内でターゲットテーブル名です。

JSON 形式のデータを DynamoDB テーブルに挿入する

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。アイテムを挿入するテーブルを決定するには、LoadTableのメソッドTableクラスのインスタンスを指定し、AmazonDynamoDBClientクラスとDynamoDB 内でターゲットテーブル名です。

DynamoDB データ型の JSON への変換

呼び出すたびに、ToJsonのメソッドDocumentクラスを作成し、結果の JSON データでFromJsonメソッドを使用して、JSON データをインスタンスに戻すことができます。Documentクラスでは、一部のDynamoDB データ型が期待どおりに変換されません。具体的には次のとおりです。

- DynamoDB セット (SS,NS, およびBS型) は、JSON の配列に変換されます。
- DynamoDB バイナリスカラーとセット (Bおよび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

DynamoDB を使用した JSON のプログラミングの詳細と例については、AWS SDK for .NET「」を参照してください。

- DynamoDB JSON Support
- Amazon DynamoDB Update - JSON, Expanded Free Tier, Flexible Scaling, Larger Items

Amazon DynamoDB を使用した ASP.NET セッション状態の管理

> Note
>
> このトピックの情報は、.NET Framework およびAWS SDK for .NETバージョン 3.3 以前です。
>
> Warning
>
> このトピックは ASP.NET に固有のものです。このトピックの情報は、必ず ASP.NET Core に適用できるとは限りません。

ASP.NET アプリケーションは、セッション状態のデータをメモリに格納する場合があります。ただし、この方法には十分な拡張性がありません。アプリケーションが複数のウェブサーバーを対象とする規模に拡大した後、セッション状態をサーバー間で共有する必要があります。一般的な対応策は、Microsoft SQL Server を使用して専用のセッション状態サーバーをセットアップすることですが、この方法にも欠点があります。他のマシンの管理が必要になり、セッション状態サーバーが単一障害点となります。また、セッション状態サーバー自体がパフォーマンスのボトルネックとなる可能性があります。

DynamoDBからのNoSQL データベースストアです。AWSを使用すると、各ウェブサーバーを対象としたセッション状態の共有に対して効果的なソリューションが実現され、これらの欠点が回避されます。

> Note
>
> 使用するソリューションに関係なく、Amazon DynamoDB では項目のサイズに制限が適用されることに注意してください。DynamoDB に保存するどのレコードも、この制限を超えることはできません。詳細については、「」を参照してください。DynamoDB での制限『Amazon DynamoDB デベロッパーガイド』

AWS SDK for .NET には、ASP.NET セッション状態プロバイダーが含まれている `AWS.SessionProvider.dll` が付属しています。また、AmazonDynamoDBSessionProviderSampleサンプルが含まれます。このサンプルは、Amazon DynamoDB をセッション状態プロバイダーとして使用する方法を示します。

ASP.NET アプリケーションでセッション状態を使用する方法の詳細については、マイクロソフトのドキュメント。

ASP.NET_SessionState テーブルの作成

アプリケーションは、起動時に、デフォルトで Amazon DynamoDB テーブルを探します。`ASP.NET_SessionState`。最初にアプリケーションを実行する前に、このテーブルを作成することをお勧めします。

ASP.NET_SessionState テーブルを作成するには

1. [テーブルの作成] を選択します。[Create Table (テーブルの作成)] ウィザードが開きます。
2. [Table name (テーブル名)] テキストボックスに「`ASP.NET_SessionState`」と入力します。
3. [Primary key (プライマリキー)] フィールドに「`SessionId`」と入力し、そのタイプを「`String`」に設定します。
4. すべてのオプションを入力したら、[Create (作成)] を選択します。

ステータスが `ASP.NET_SessionState` から `CREATING` に変わったら、`ACTIVE` テーブルは使用できる状態になります。

> Note
>
> テーブルを事前に作成しない場合、初期化時にセッション状態プロバイダーによってテーブルが作成されます。セッション状態テーブルの設定パラメーターとして機能する属性のリストについては、下記の `web.config` オプションを参照してください。プロバイダーがテーブルを作成する場合は、これらのパラメータを使用します。

セッション状態プロバイダーの設定

DynamoDB をセッション状態サーバーとして使用するように ASP.NET アプリケーションを設定するには

1. Visual Studio ASP.NET プロジェクトに、`AWSSDK.dll` と `AWS.SessionProvider.dll` への参照を追加します。これらのアセンブリは、NuGet パッケージ (p. 29)またはアセンブリの手動インストール (p. 30)。

SDK の以前のバージョンでは、セッション状態プロバイダーの機能は `AWS.Extension.dll` に含まれていました。使いやすさを向上させるために、この機能は `AWS.SessionProvider.dll` に移動されました。詳細については、ブログの投稿記事を参照してください。AWS.Extension名前変更中。

2. アプリケーションの `Web.config` ファイルを編集します。`system.web` 要素で、既存の `sessionState` 要素を次の XML フラグメントに置き換えます。

```
<sessionState timeout="20" mode="Custom" customProvider="DynamoDBSessionStoreProvider">
  <providers>
    <add name="DynamoDBSessionStoreProvider"
         type="Amazon.SessionProvider.DynamoDBSessionStateStore"
         AWSProfileName="{profile_name}"
         Region="us-west-2" />
  </providers>
</sessionState>
```

プロファイルは、AWS認証情報。この認証情報は、DynamoDB との通信で使用され、セッション状態の保存と取得に使用されます。使用している場合、AWS SDK for .NETでプロファイルを指定しているappSettingsセクションのWeb.configファイルでプロファイルを指定する必要はありません。providersセクションに追加します。AWS.NETクライアントコードが実行時に検出します。詳細については、「」を参照してください。の設定AWS SDK for .NETアプリケーション (p. 17)。

ウェブサーバーが、EC2 インスタンスの IAM ロールを使用するように設定されている Amazon EC2 インスタンスで実行されている場合は、`Web.config`ファイルを開きます。この場合は、AWS.NET クライアントは、IAM ロール認証情報を使用します。詳細については、「IAM ロールを使用したアクセス権限の付与 (p. 151)」と「セキュリティに関する考慮事項 (p. 91)」を参照してください。

Web.config オプション

`providers` ファイルの `Web.config` セクションで、次の設定属性を使用できます。

AWSAccessKey

使用するアクセスキー ID。これは、`providers` セクションまたは `appSettings` セクションで設定できます。この設定を使用しないことをお勧めします。代わりに、`AWSProfileName` を使用して認証情報を指定し、プロファイルを指定してください。

AWSSecretKey

使用するシークレットキー。これは、`providers` セクションまたは `appSettings` セクションで設定できます。この設定を使用しないことをお勧めします。代わりに、`AWSProfileName` を使用して認証情報を指定し、プロファイルを指定してください。

AWSProfileName

使用する認証情報に関連付けられるプロファイルの名前。詳細については、「」を参照してください。の設定AWS SDK for .NETアプリケーション (p. 17)。

リージョン

必須の `string` 属性。-AWSAmazon DynamoDB を使用するリージョンです。のリストについてはAWSリージョンについては、」リージョンとエンドポイント: DynamoDB。

アプリケーション

オプションの `string` 属性。`Application` 属性の値は、テーブルのセッションデータをパーティション分割するために使用されます。これにより、テーブルを複数のアプリケーションで使用することができます。

表

オプションの `string` 属性。セッションデータの格納に使用されるテーブルの名前。デフォルト: `ASP.NET_SessionState`。

ReadCapacityUnits

オプションの `int` 属性。プロバイダーがテーブルを作成する際に使用する読み取りキャパシティーユニット。デフォルトは 10 です。

WriteCapacityUnits

オプションの `int` 属性。プロバイダーがテーブルを作成する際に使用する書き込みキャパシティーユニット。デフォルトは 5 です。

CreateIfNotExist

オプションの `boolean` 属性。`CreateIfNotExist` 属性は、テーブルが存在しない場合にプロバイダーがテーブルを自動作成するかどうかを制御します。デフォルトは true です。このフラグが false に設定されていて、テーブルが存在しない場合は、例外がスローされます。

セキュリティに関する考慮事項

DynamoDB テーブルが作成され、アプリケーションが設定されると、どのセッションプロバイダーでもセッションを使用できるようになります。

セキュリティのベストプラクティスとして、IAM ユーザーの認証情報を使用してアプリケーションを実行することをお勧めします。以下のいずれかを使用できます。IAM マネジメントコンソールまたはAWS Toolkit for Visual Studioを使用して IAM ユーザーを作成し、アクセスポリシーを定義します。

セッション状態プロバイダーでは、セッションデータを保存するテーブルに対して、DeleteItem、DescribeTable、GetItem、PutItem、UpdateItem、の各オペレーションの呼び出しが可能になっている必要があります。次のサンプルポリシーを使用して、us-west-2 で実行している DynamoDB のインスタンスのプロバイダーが必要とするオペレーションだけを使用できるように、IAM ユーザーを制限することができます。

```
{ "Version" : "2012-10-17",
"Statement" : [
  {
    "Sid" : "1",
    "Effect" : "Allow",
    "Action" : [
        "dynamodb:DeleteItem",
        "dynamodb:DescribeTable",
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:UpdateItem"
    ],
    "Resource" : "arn:aws:dynamodb:us-west-2:{<YOUR-AWS-ACCOUNT-ID>}:table/
ASP.NET_SessionState"
    }
  ]
}
```

Amazon EC2 での作業

-AWS SDK for .NETが をサポートAmazon EC2。これは、サイズ変更可能なコンピューティング能力を提供するWebサービスです。このコンピューティング能力を使用して、ソフトウェアシステムの構築とホストを行います。

APIs

-AWS SDK for .NETは Amazon EC2 クライアント用 API を提供します。API を使用すると、セキュリティグループやキーペアなどの EC2 機能を操作できます。API を使用すると、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 を操作するときに従うことができるパターンを示す少数の例を紹介します。API の完全なセットを表示するには、を参照してください。AWS SDK for .NETAPI リファレンス(「Amazon.ec2」までスクロールします)。

Amazon EC2 API は、AWSSDK.EC2NuGet パッケージです。

Prerequisites

始める前に、「」環境をセットアップする (p. 15)。また、「」の情報を確認します。プロジェクトをセットアップする (p. 17)およびSDK の機能 (p. 49)。

例について

このセクションの例では、Amazon EC2 クライアントを操作し、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 (p. 125)に、Amazon EC2 スポットインスタンスをリクエストする方法を示します。スポットインスタンスを使用すると、オンデマンド価格より低価で未使用の EC2 キャパシティにアクセスできます。

トピック

- Amazon EC2 でのセキュリティグループの使用 (p. 92)
- Amazon EC2 のキーペアでの作業 (p. 105)
- Amazon EC2 リージョンとアベイラビリティーゾーンの情報を表示する (p. 112)
- Amazon EC2 インスタンスの使用 (p. 113)
- Amazon EC2 スポットインスタンスのチュートリアル (p. 125)

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と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セキュリティグループを作成できます。たとえば、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、Linux 用 EC2 ユーザーガイドとWindows 用 EC2 ユーザーガイド。

を使用する場合AWS SDK for .NETでは、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、「Linux 用 EC2 ユーザーガイドまたはWindows 用 EC2 ユーザーガイド。

API と前提条件の詳細については、親セクション (Amazon EC2 での作業 (p. 91)).

トピック

- セキュリティグループの列挙 (p. 92)
- セキュリティグループの作成 (p. 95)
- セキュリティグループの更新 (p. 101)

セキュリティグループの列挙

この例では、以下の方法を示します。AWS SDK for .NETをクリックしてセキュリティグループを列挙します。あなたが提供する場合Amazon Virtual Private CloudID の場合、アプリケーションはその特定の VPC

のセキュリティグループを列挙します。それ以外の場合、アプリケーションは使用可能なすべてのセキュリティグループのリストを表示するだけです。

次のセクションでは、この例のスニペットを学習できます。-例の完全なコード (p. 93)を後に示し、そのままビルドして実行できる。

トピック

- セキュリティグループを列挙する (p. 93)
- コードの完了 (p. 93)
- 追加の考慮事項 (p. 95)

セキュリティグループを列挙する

次のスニペットは、セキュリティグループを列挙しています。特定の VPC のすべてのグループまたはグループが指定されている場合は、そのグループを列挙します。

例このトピックの最後にある (p. 93)に、このスニペットが使用中であることを示します。

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

コードの完了

このセクションでは、この例の関連するリファレンスと、完全なコードを示します。

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

  クラスセキュリティグループリクエストについて説明する

  クラスDescribeSecurityGroupsResponse

  クラスフィルタ

  クラスSecurityGroup

ザ・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、フィルターは`Name`「vpc-id」に設定された名前と値のペアの一部。この名前は、`Filters`の財産セキュリティグループリクエストについて説明するクラス。

- セキュリティグループの完全なリストを取得するには、SecurityGroupsAsync をパラメータなしで記述します。。

- でセキュリティグループのリストを確認することで、結果を確認できます。Amazon EC2 コンソール。

セキュリティグループの作成

この例では、以下の方法を示します。AWS SDK for .NETをクリックしてセキュリティグループを作成します。既存の VPC の ID を指定して、VPC に EC2 のセキュリティグループを作成できます。そのようなIDを指定しないと、新しいセキュリティグループが EC2-Classic の場合は、AWSアカウントはこれをサポートしています。

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の VPC に属します。詳細については、VPC の EC2 のリファレンスと EC2-Classic の親セクション () を参照してください。Amazon EC2 でのセキュリティグループの使用 (p. 92)).

次のセクションでは、この例のスニペットを学習できます。-例の完全なコード (p. 97)を後に示し、そのままビルドして実行できる。

トピック

既存のセキュリティグループの検索

次のスニペットは、指定された VPC 内の特定の名前の既存のセキュリティグループを検索します。

例このトピックの最後にある (p. 97)に、このスニペットが使用中であることを示します。

```
//
// 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

次のスニペットは、その名前のグループが指定された VPC に存在しない場合、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、その名前のグループが 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。

例このトピックの最後にある (p. 97)に、このスニペットが使用中であることを示します。

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

コードの完了

このセクションでは、この例の関連するリファレンスと、完全なコードを示します。

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

  クラスセキュリティグループリクエストについて説明する

  クラスDescribeSecurityGroupsResponse

  クラスフィルタ

  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、以下の方法を示します。AWS SDK for .NETをクリックして、セキュリティグループにルールを追加します。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。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。「」を参照してください。追加の考慮事項このトピックのこの最後で、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説明します。

次のセクションでは、この例のスニペットを学習できます。-例の完全なコード (p. 101)を後に示し、そのままビルドして実行できる。

トピック

- インバウンドルールの追加 (p. 101)
- コードの完了 (p. 101)
- 追加の考慮事項 (p. 104)

インバウンドルールの追加

次のスニペットは、特定の IP アドレス (または範囲) および TCP ポートの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。

例このトピックの最後にある (p. 101)に、このスニペットが使用中であることを示します。

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

コードの完了

このセクションでは、この例の関連するリファレンスと、完全なコードを示します。

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、アプリケーションのデフォルトはポート 3389 になります。これは Windows RDP のポートで、Windows を実行している EC2 インスタンスに接続できます。Linux を実行する 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。

- 例がセットされていることに注目してください。`IpProtocol`「tcp」へ。の値`IpProtocol`の記述にあります。`IpProtocol`の財産IpPermissionクラス。

- この例を使用する場合は、ローカルコンピュータの IP アドレスを指定する必要があります。住所を取得する方法のいくつかを以下に示します。

- EC2 インスタンスに接続するローカルコンピューターに静的パブリック IP アドレスがある場合は、サービスを使用してそのアドレスを取得できます。そのようなサービスの一つはhttp://checkip.amazonaws.com/。インバウンドトラフィックの承認の詳細については、Linux 用 EC2 ユーザーガイドまたはWindows 用 EC2 ユーザーガイド。
- ローカルコンピュータの IP アドレスを取得するもう 1 つの方法は、Amazon EC2 コンソール。

  いずれかのセキュリティグループを選択し、インバウンドルールタブをクリックし、インバウンドのルールの編集。インバウンドルールで、ドロップダウンメニューを送信元列を選択し、マイ IPをクリックして、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。次のことを確認してください。キャンセルオペレーションの例。

- この例の結果を確認するには、でセキュリティグループのリストを調べます。Amazon EC2 コンソール。

Amazon EC2 のキーペアでの作業

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は、起動時にkey pair を指定し、接続するときはペアのプライベートキーを提供する必要があります。

EC2 インスタンスを起動するときは、key pair を作成することも、他のインスタンスの起動時にすでに使用したキーペアを使用することもできます。Amazon EC2 のキーペアの詳細については、Linux 用 EC2 ユーザーガイドまたはWindows 用 EC2 ユーザーガイド。

API と前提条件の詳細については、親セクション (Amazon EC2 での作業 (p. 91)).

トピック

- キーペアの作成と表示 (p. 105)
- キーペアの削除 (p. 109)

キーペアの作成と表示

この例では、以下の方法を示します。AWS SDK for .NETをクリックして、key pair を作成します。アプリケーションは、新しいkey pair 名前と PEM ファイルの名前 (拡張子は「.pem」) を取ります。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使用可能なすべてのキーペアを表示します。

次のセクションでは、この例のスニペットを学習できます。-例の完全なコード (p. 106)を後に示し、そのままビルドして実行できる。

トピック

- key pair 作成 (p. 105)
- 使用可能なキーペアを表示する (p. 106)
- コードの完了 (p. 106)
- 追加の考慮事項 (p. 109)

key pair 作成

次のスニペットは、key pair を作成し、プライベートキーを指定された PEM ファイルに格納します。

例このトピックの最後にある (p. 106)に、このスニペットが使用中であることを示します。

```
//
// 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を表示する

次のスニペットでは、利用可能なキーペアのリストを表示します。

例このトピックの最後にある (p. 106)に、このスニペットが使用中であることを示します。

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

コードの完了

このセクションでは、この例の関連するリファレンスと、完全なコードを示します。

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

- この例の実行後、新しいkey pair をAmazon EC2 コンソール。

- key pair を作成するときは、後でプライベートキーを取得できないので、返されたプライベートキーを保存する必要があります。

キーペアの削除

この例では、以下の方法を示します。AWS SDK for .NETをクリックして、key pair を削除します。アプリケーションは、key pair 名前を取ります。key pair が削除され、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使用可能なすべてのキーペアを表示します。

次のセクションでは、この例のスニペットを学習できます。-例の完全なコード (p. 110)を後に示し、そのままビルドして実行できる。

トピック

key pair を削除する

以下のスニペットは、key pair を削除します。

例このトピックの最後にある (p. 110)に、このスニペットが使用中であることを示します。

```
//
// 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を表示する

次のスニペットでは、利用可能なキーペアのリストを表示します。

例このトピックの最後にある (p. 110)に、このスニペットが使用中であることを示します。

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

コードの完了

このセクションでは、この例の関連するリファレンスと、完全なコードを示します。

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 リージョンとアベイラビリティーゾーンの情報を表示する

Amazon EC2 は、世界各地の口ケーションでホスティングされています。これらの口ケーションは、 リージョンとアベイラビリティーゾーンから構成されています。各リージョンは地理的に離れた地域であり、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情報を参照してくださいLinux 用 EC2 ユーザーガイドまたはWindows 用 EC2 ユーザーガイド。

この例では、以下の方法を示します。AWS SDK for .NETEC2 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を取得します。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

  クラスリージョン

```
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

♪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、これを行う方法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、Linux 用 EC2 ユーザーガイドまたはWindows 用 EC2 ユーザーガイド

API と前提条件の詳細については、親セクション (Amazon EC2 での作業 (p. 91)).

トピック

- Amazon EC2 インスタンスの起動 (p. 113)
- Amazon EC2 インスタンスの削除 (p. 124)

Amazon EC2 インスタンスの起動

この例では、以下の方法を示します。AWS SDK for .NET同じ Amazon Machine Image (AMI) からまったく同じに設定された 1 つ以上の Amazon EC2 インスタンスを起動できます。を使用する複数の入力 (p. 114)指定すると、アプリケーションは EC2 インスタンスを起動し、インスタンスが「Pending」状態に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している場合は、「」の説明に従って、リモートで接続できます。(オプション) インスタンスConnect。 (p. 121)。

EC2 インスタンスは VPC または EC2-Classic で起動できます (AWSアカウントはこれをサポートしています)。VPC 内の EC2 と EC2-Classic の詳細については、「Linux 用 EC2 ユーザーガイドまたはWindows 用 EC2 ユーザーガイド。

次のセクションでは、この例のスニペットおよびその他の情報について説明します。-例の完全なコード (p. 116)はスニペットの後に表示され、そのままビルドして実行できます。

トピック

- 必要なものを集める (p. 114)
- インスタンスの起動 (p. 114)
- インスタンスのモニタリング (p. 115)
- コードの完了 (p. 116)

- 追加の考慮事項 (p. 120)
- (オプション) インスタンスConnect。 (p. 121)
- クリーンアップ (p. 124)

必要なものを集める

EC2 インスタンスを起動するには、いくつかのものが必要です。

- AVPCインスタンスの起動場所。Windows インスタンスで RDP 経由で接続する場合は、ほとんどの場合、VPC にはインターネットゲートウェイと、ルートテーブル内のインターネットゲートウェイのエントリがアタッチされている必要があります。詳細については、「」を参照してください。インターネットゲートウェイのAmazon VPC User Guide。

- インスタンスを起動する VPC 内の既存のサブネットの ID。これを見つけたり作成したりする簡単な方法は、にサインインすることです。Amazon VPC コンソールが成り立っている。ただし、プログラムによっても取得できる。CreateSubnetAsyncおよびSubnetsaSyncを記述するメソッド。

  Note

  あなたのAWSアカウントは EC2-Classic をサポートしており、起動するインスタンスのタイプです。このパラメータは必須ではありません。ただし、アカウントが EC2-Classic をサポートせず、このパラメータを指定しない場合、新しいインスタンスがアカウントのデフォルト VPC で起動されます。

- インスタンスを起動する VPC に属する既存のセキュリティグループの ID。詳細については、「Amazon EC2 でのセキュリティグループの使用 (p. 92)」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 (Linux インスタンス) で SSH トラフィックを許可するか、ポート 3389 (Windows インスタンス) の RDP トラフィックを許可する適切なインバウンドルールが必要です。これを行う方法については、を参照してください。セキュリティグループの更新 (p. 101)を含む。追加の考慮事項 (p. 104)そのトピックの最後近くに。

- インスタンスの作成に使用される Amazon Machine Image (AMI) です。AMI に関する情報については、を参照してください。Linux 用 EC2 ユーザーガイドまたはWindows 用 EC2 ユーザーガイド。たとえば、共有 AMI については、Linux 用 EC2 ユーザーガイドまたはWindows 用 EC2 ユーザーガイド。

- 新しいインスタンスへの接続に使用される既存の EC2 key pair 名前。詳細については、「Amazon EC2 のキーペアでの作業 (p. 105)」を参照してください。

- 前述の EC2 キーkey pair 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、次の場合に使用されます。リモート接続 (p. 121)インスタンスへの。

インスタンスの起動

次のスニペットでは、EC2 インスタンスを起動します。

例このトピックの最後にある (p. 116)に、このスニペットが使用中であることを示します。

```
//
// 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

次のスニペットは、インスタンスが「Pending」状態になるまでインスタンスを監視します。

例このトピックの最後にある (p. 116)に、このスニペットが使用中であることを示します。

フレームワークの使用の詳細については、InstanceStateの有効な値のクラス`Instance.State.Code`プロパティ。

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

コードの完了

このセクションでは、この例の関連するリファレンスと、完全なコードを示します。

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

  クラスインスタンス

  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、フィルタを`Filter`の財産DescribeInstancesRequestオブジェクト。この方法を使用すると、特定のユーザー指定タグを持つインスタンスなど、特定のインスタンスへのリクエストを制限できます。

- 簡潔にするために、いくつかのプロパティに典型的な値が与えられました。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決定できます。

- に使用できる値は`MinCount`および`MaxCount`のプロパティRunInstancesRequestオブジェクトは、ターゲットのアベイラビリティーゾーンと、インスタンスタイプに対して許可されているインスタンスの最大数によって決まります。詳細については、「」を参照してください。Amazon EC2 で実行できるインスタンスの数はいくつですかAmazon EC2 全般に関するよくある質問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を選択できます。このインスタンスタイプは、Linux 用 EC2 ユーザーガイドまたはWindows 用 EC2 ユーザーガイド。

- また、IAM ロール (p. 151)インスタンスの起動時に、インスタンスにします。そうするためには、IamInstanceProfileSpecificationそのオブジェクトNameプロパティに IAM ロールの名前が設定されています。次に、そのオブジェクトをIamInstanceProfileの財産RunInstancesRequestオブジェクト。

  Note

  IAM ロールがアタッチされた 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、「」を参照してください。Linux 用 EC2 ユーザーガイドまたはWindows 用 EC2 ユーザーガイド。

(オプション) インスタンスConnect。

インスタンスを実行した後で、適切なリモートクライアントを使用してリモート接続できます。Linux インスタンスと Windows インスタンスの両方で、インスタンスのパブリック IP アドレスまたはパブリック DNS 名が必要です。また、以下も必要です。

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。「」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が、ポート 22 の SSH トラフィックを許可していることを確認します。セキュリティグループの更新 (p. 101)。

インスタンスの起動に使用したkey pair プライベート部分、つまり PEM ファイルが必要です。

詳細については、「」を参照してください。Linux インスタンスへの接続Linux インスタンス用 Amazon EC2 ユーザーガイド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インスタンスに接続できます。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が、ポート 3389 で RDP トラフィックを許可していることを確認します。セキュリティグループの更新 (p. 101)。

管理者パスワードも必要です。これを取得するには、次のコード例を使用します。このコードには、インスタンス ID と、インスタンスの起動に使用されたkey pair プライベート部分、つまり PEM ファイルが必要です。

詳細については、「」を参照してください。Windows インスタンスへの接続Windows インスタンス用 Amazon EC2 ユーザーガイドを参照してください。

Warning

このサンプルコードは、インスタンスのプレーンテキストの管理者パスワードを返します。

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GetPasswordData Request

クラスGetPasswordData Response

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 インスタンスが必要なくなったときは、「」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 インスタンスの削除 (p. 124)。

Amazon EC2 インスタンスの削除

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、以下の方法を示します。AWS SDK for .NETEC2 インスタンスを終了します。インスタンス ID を入力として受け入れます。

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);
      }
    }
  }
}
```

この例の実行後は、にサインインすることをお勧めします。Amazon EC2 コンソール次のことを確認して、EC2 インスタンスは終了しました。

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、以下の方法を示します。AWS SDK for .NETをクリックして Amazon EC2 スポットインスタンスを管理します。

Overview

スポットインスタンスを使用すると、オンデマンド価格より低価で未使用の Amazon EC2 キャパシティーをリクエストできます。これにより、中断される可能性のある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の概要は次のとおりです。

1. スポットインスタンスリクエストを作成します。このようにすれば、支払える価格の上限を指定します。
2. リクエストが満たされたら、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. 必要な期間インスタンスを実行し、終了する。スポット価格インスタンスが自動的に終了するように変更されます。
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作成されなくなります。

これは、スポットインスタンスの非常に高いレベルの概要です。スポットインスタンスについては、「」でスポットインスタンスについて理解を深めることができます。Linux 用 EC2 ユーザーガイドまたはWindows 用 EC2 ユーザーガイド。

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、AWS SDK for .NET] をクリックして、次の作業を行います。

- 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

次のセクションでは、この例のスニペットおよびその他の情報について説明します。-例の完全なコード (p. 130)はスニペットの後に表示され、そのままビルドして実行できます。

トピック

Prerequisites

API と前提条件の詳細については、親セクション (Amazon EC2 での作業 (p. 91)).

必要なものを集める

スポットインスタンスリクエストを作成するには、いくつかのものが必要です。

- 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にはさまざまなものがあります。Linux 用 EC2 ユーザーガイドまたはWindows 用 EC2 ユーザーガイド。このチュートリアルのデフォルトの数値は 1 です。

- インスタンスの作成に使用される Amazon Machine Image (AMI) です。AMI に関する情報については、を参照してください。Linux 用 EC2 ユーザーガイドまたはWindows 用 EC2 ユーザーガイド。たとえば、共有 AMI については、Linux 用 EC2 ユーザーガイドまたはWindows 用 EC2 ユーザーガイド。

- インスタンス時間あたりに支払うことができる上限料金。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) の価格は、Amazon EC2 の料金表。このチュートリアルのデフォルト価格については、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。詳細については、で説明しています。Amazon EC2 でのセキュリティグループの使用 (p. 92)に関する情報は次のとおりです。必要なものを集める (p. 114)およびインスタンスへの接続 (p. 121)がAmazon EC2 インスタンスの起動 (p. 113)。簡単にするために、このチュートリアルでは、という名前のセキュリティグループを使用します。defaultすべての新しいことAWSアカウントにはがあります。

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。の一般的な戦略は以下のとおりです。

- オンデマンド料金を下回るリクエストを作成します。
- 計算結果の値に基づいてリクエストを行います。
- コンピューティングキャパシティーをできるだけ早く取得するようにリクエストする。

以下の説明は、「スポット価格」の履歴を参照してください。Linux 用 EC2 ユーザーガイドまたはWindows 用 EC2 ユーザーガイド。

コストをオンデマンドよりも低くする

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。

インスタンスタイプのスポット料金履歴は、Amazon EC2 コンソールまたは Amazon EC2 API を使用して調べます。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、オンデマンド価格より下) でリクエストを指定します。このようにすれば、この 1 回限りのスポットインスタンスリクエストが受理される可能性が高くなり、ジョブが完了するまで連続して実行できるからです。
- 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。

結果の価値を超えないように支払う

データ処理ジョブを実行するとします。このジョブの結果の価値はわかっており、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後で、価格を選択します。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がないように、価格を決定します。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キャパシティを迅速に獲得

オンデマンドインスタンスでは追加キャパシティーが突然、短期間だけ必要になることがあり、このようなキャパシティを獲得できないとします。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後で、履歴の価格の最大値を超える価格を選択します。このようにすれば、リクエストがすぐに受理される可能性が高まり、計算が完了するまで連続して計算できるようになります。

必要なもの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できます。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 (このチュートリアルでは 0.003

USD) と同額に設定します。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を作成する

次のスニペットは、前に収集し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。

例このトピックの最後にある (p. 130)に、このスニペットが使用中であることを示します。

```
//
// Method to create a Spot Instance request
private static async Task<SpotInstanceRequest> CreateSpotInstanceRequest(
  IAmazonEC2 ec2Client, string amiId, string securityGroupName,
  InstanceType instanceType, string spotPrice, int instanceCount)
{
  var launchSpecification = new LaunchSpecification{
    ImageId = amiId,
    InstanceType = instanceType
  };
  launchSpecification.SecurityGroups.Add(securityGroupName);
  var request = new RequestSpotInstancesRequest{
    SpotPrice = spotPrice,
    InstanceCount = instanceCount,
    LaunchSpecification = launchSpecification
  };

  RequestSpotInstancesResponse result =
    await ec2Client.RequestSpotInstancesAsync(request);
  return result.SpotInstanceRequests[0];
}
```

このメソッドから返される重要な値は、スポットインスタンスリクエスト ID で、`SpotInstanceRequestId`返品されたのメンバーSpotInstanceRequestオブジェクト。

> **Note**
>
> 起動されたスポットインスタンスに対して課金されます。不必要なコストを回避するには、リクエストをキャンセルする (p. 129)およびインスタンスを削除する (p. 129)。

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満たされるのを待つかどうかなど、コード内で特定の決定を下すことができます。

例このトピックの最後にある (p. 130)に、このスニペットが使用中であることを示します。

```
//
// 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];
```

```
    }
```

このメソッドは、インスタンス ID、状態、ステータスコードなど、スポットインスタンスリクエストに関する情報を返します。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、Linux 用 EC2 ユーザーガイドまたはWindows 用 EC2 ユーザーガイド。

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は、未処理のリクエストをキャンセルして、それらのリクエストが再実行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ます。

例このトピックの最後にある (p. 130)に、このスニペットが使用中であることを示します。

```
    //
    // 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開始されたインスタンスを終了することが重要です。単にスポットインスタンスリクエストをキャンセルしてもインスタンスリクエストは終了しないので、引き続き課金されます。次のスニペットは、アクティブなスポットインスタンスのインスタンス ID を取得した後で、インスタンスを終了する方法を示しています。

例このトピックの最後にある (p. 130)に、このスニペットが使用中であることを示します。

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

```
    }
```

コードの完了

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PotInstance リクエストのレスポンスを説明する

  クラスInstanceStateChange

  クラスLaunchSpecification

  クラスSpotInstancesRequest

  クラスSpotInstancesResponse のリクエスト

  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後は、にサインインすることをお勧めします。Amazon EC2 コンソール次のことを確認して、スポットインスタンスリクエストがキャンセルされ、スポットインスタンスは終了しました。

へのアクセスAWS Identity and Access Managementと (IAM)AWS SDK for .NET

-AWS SDK for .NETが をサポートAWS Identity and Access Management。これは、有効にするウェブサービスですAWSカスタマーがでユーザーとユーザーアクセス許可を管理するAWS。

AnAWS Identity and Access Management(IAM)ユーザーは、で作成したエンティティですAWS。エンティティは、とやり取りするユーザーまたはアプリケーションを表します。AWS。IAM ユーザーの詳細については、「」を参照してください。IAM ユーザーおよびIAM および STS の制限IAM ユーザーガイド

IAM 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。ポリシーにはポリシードキュメントユーザーが実行できるアクションと、そのアクションが影響を与えるリソースの一覧が記載されています。IAM ポリシーの詳細については、「」を参照してください。ポリシーとアクセス許可のIAM ユーザーガイド。

APIs

-AWS SDK for .NETIAM クライアント用の API 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 機能を操作できます。

このセクションでは、これらの API を操作するときに従うことができるパターンを示す少数の例を紹介します。API の完全なセットを表示するには、を参照してください。AWS SDK for .NETAPI リファレンス(「Amazon.IdentityManagement」までスクロールします)。

このセクションには次のものも含まれます。の例 (p. 151)これは、IAM ロールを Amazon EC2 インスタンスにアタッチして、認証情報の管理を容易にする方法を示しています。

IAM API は、awssdk.IDENTITY 管理NuGet パッケージです。

Prerequisites

開始する前に、必ず環境をセットアップする (p. 15)。また、「」の情報を確認します。プロジェクトをセットアップする (p. 17)およびSDK の機能 (p. 49)。

Topics

トピック

- のユーザーの作成とリスト表示AWSアカウント (p. 134)
- IAM ユーザーの削除 (p. 140)
- JSON から IAM 管理ポリシーを作成する (p. 144)
- IAM 管理ポリシーのポリシードキュメントを表示する (p. 148)
- IAM ロールを使用してアクセスを許可する (p. 151)

のユーザーの作成とリスト表示AWSアカウント

この例では、AWS SDK for .NETをクリックして、IAM ユーザーを作成します。アプリケーションに指定した情報を使用して、ユーザーを作成し、指定された管理ポリシーをアタッチし、そのユーザーの資格情報を取得し、内のすべてのユーザーのリストを表示します。AWSアカウント.

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すべてのユーザーのリストを表示します。AWSアカウント.

指定する入力の 1 つは、既存の管理ポリシーの Amazon リソースネーム (ARN) です。利用可能なポリシーとその ARN は、IAM コンソール。

ここで示している各セクションで、この例のスニペットを学習できます。-例のための完全なコード (p. 136)を後に示し、そのままビルドして実行できる。

トピック

- ユーザーの作成 (p. 134)
- ユーザーのリストを表示する (p. 135)
- コードの完了 (p. 136)
- 追加の考慮事項 (p. 140)

ユーザーの作成

次のスニペットは、IAM ユーザーを作成し、指定された管理セキュリティポリシーを追加し、そのユーザーの認証情報を作成して保存します。

の例このトピックの最後にある (p. 136)に、このスニペットが使用中であることを示します。

```
//
// Method to create the user
private static async Task<CreateUserResponse> CreateUser(
  IAmazonIdentityManagementService iamClient, string userName,
  string policyArn, string csvFilename)
{
  // Create the user
  // Could also create a login profile for the user by using CreateLoginProfileAsync
  CreateUserResponse responseCreate =
    await iamClient.CreateUserAsync(new CreateUserRequest(userName));
```

```
  // Attach an existing managed policy
  await iamClient.AttachUserPolicyAsync(new AttachUserPolicyRequest{
    UserName = responseCreate.User.UserName,
    PolicyArn = policyArn});

  // Create credentials and write them to a CSV file.
  CreateAccessKeyResponse responseCreds =
    await iamClient.CreateAccessKeyAsync(new CreateAccessKeyRequest{
      UserName = responseCreate.User.UserName});
  using (FileStream s = new FileStream(csvFilename, FileMode.Create))
  using (StreamWriter writer = new StreamWriter(s))
  {
    writer.WriteLine("User name,Access key ID,Secret access key");
    writer.WriteLine("{0},{1},{2}", responseCreds.AccessKey.UserName,
      responseCreds.AccessKey.AccessKeyId,
      responseCreds.AccessKey.SecretAccessKey);
  }

  return responseCreate;
}
```

ユーザーのリストを表示する

次のスニペットは、既存のユーザーのリストと、アクセスキー ID やアタッチされたポリシーなどの各ユーザーに関する情報を表示します。

の例このトピックの最後にある (p. 136)に、このスニペットが使用中であることを示します。

```
//
// Method to print out a list of the existing users and information about them
private static async Task ListUsers(IAmazonIdentityManagementService iamClient)
{
  // Get the list of users
  ListUsersResponse responseUsers = await iamClient.ListUsersAsync();
  Console.WriteLine("\nFull list of users...");
  foreach (var user in responseUsers.Users)
  {
    Console.WriteLine($"User {user.UserName}:");
    Console.WriteLine($"\tCreated: {user.CreateDate.ToShortDateString()}");

    // Show the list of groups this user is part of
    ListGroupsForUserResponse responseGroups =
      await iamClient.ListGroupsForUserAsync(
        new ListGroupsForUserRequest(user.UserName));
    foreach (var group in responseGroups.Groups)
      Console.WriteLine($"\tGroup: {group.GroupName}");

    // Show the list of access keys for this user
    ListAccessKeysResponse responseAccessKeys =
      await iamClient.ListAccessKeysAsync(
        new ListAccessKeysRequest{UserName = user.UserName});
    foreach(AccessKeyMetadata accessKey in responseAccessKeys.AccessKeyMetadata)
      Console.WriteLine($"\tAccess key ID: {accessKey.AccessKeyId}");

    // Show the list of managed policies attached to this user
    var requestManagedPolicies = new ListAttachedUserPoliciesRequest{
      UserName = user.UserName};
    ListAttachedUserPoliciesResponse responseManagedPolicies =
      await iamClient.ListAttachedUserPoliciesAsync(
        new ListAttachedUserPoliciesRequest{UserName = user.UserName});
    foreach(var policy in responseManagedPolicies.AttachedPolicies)
      Console.WriteLine($"\tManaged policy name: {policy.PolicyName}");
```

```
        // Show the list of inline policies attached to this user
        ListUserPoliciesResponse responseInlinePolicies =
          await iamClient.ListUserPoliciesAsync(
            new ListUserPoliciesRequest(user.UserName));
        foreach(var policy in responseInlinePolicies.PolicyNames)
          Console.WriteLine($"\tInline policy name: {policy}");
      }
    }
```

コードの完了

このセクションでは、関連する参考資料と、この例の完全なコードを示します。

SDK リファレンス

NuGet パッケージ:

- awssdk.IDENTITY 管理

プログラミング要素:

- 名前空間Amazon.IDENTITY 管理

  クラスAmazonIdentityManagementServiceClient
- 名前空間Amazon.identityManagement.model

  クラスユーザーポリシーリクエストをアタッチ

  クラスCreateAccessKeyRequest

  クラスアクセスキーレスポンスの作成

  クラスCreateUserRequest

  クラスユーザーレスポンスの作成

  クラスListAccessKeysRequest

  クラスListAccessKeysResponse

  クラスリストされたユーザーポリシーのリクエスト

  クラスリストされたユーザーポリシーレスポンス

  クラスユーザーリクエストのグループを一覧表示

  クラスユーザーレスポンスのグループを一覧表示

  クラスリストユーザーポリシーのリクエスト

  クラスリストユーザーポリシーレスポンス

  クラスListUsersResponse

コード

```
using System;
using System.Collections.Generic;
using System.IO;
```

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

namespace IamCreateUser
{
  // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
 =
  // Class to create a user
  class Program
  {
    private const int MaxArgs = 3;

    static async Task Main(string[] args)
    {
      // Create an IAM service client
      var iamClient = new AmazonIdentityManagementServiceClient();

      // Parse the command line and show help if necessary
      var parsedArgs = CommandLine.Parse(args);
      if((parsedArgs.Count == 0) || (parsedArgs.Count > MaxArgs))
      {
        PrintHelp();
        Console.WriteLine("\nIncorrect number of arguments specified.");
        Console.Write("Do you want to see a list of the existing users? ((y) or n): ");
        string response = Console.ReadLine();
        if((string.IsNullOrEmpty(response)) || (response.ToLower() == "y"))
          await ListUsers(iamClient);
        return;
      }

      // Get the application arguments from the parsed list
      string userName =
        CommandLine.GetArgument(parsedArgs, null, "-u", "--user-name");
      string policyArn =
        CommandLine.GetArgument(parsedArgs, null, "-p", "--policy-arn");
      string csvFilename =
        CommandLine.GetArgument(parsedArgs, null, "-c", "--csv-filename");
      if(   (string.IsNullOrEmpty(policyArn) || !policyArn.StartsWith("arn:"))
         || (string.IsNullOrEmpty(csvFilename) || !csvFilename.EndsWith(".csv"))
         || (string.IsNullOrEmpty(userName)))
        CommandLine.ErrorExit(
          "\nOne or more of the required arguments is missing or incorrect." +
          "\nRun the command with no arguments to see help.");

      // Create a user, attach a managed policy, and obtain credentials
      var responseCreate =
        await CreateUser(iamClient, userName, policyArn, csvFilename);
      Console.WriteLine($"\nUser {responseCreate.User.UserName} was created.");
      Console.WriteLine($"User ID: {responseCreate.User.UserId}");

      // Output a list of the existing users
      await ListUsers(iamClient);
    }


    //
    // Method to create the user
    private static async Task<CreateUserResponse> CreateUser(
      IAmazonIdentityManagementService iamClient, string userName,
      string policyArn, string csvFilename)
    {
      // Create the user
      // Could also create a login profile for the user by using CreateLoginProfileAsync
      CreateUserResponse responseCreate =
        await iamClient.CreateUserAsync(new CreateUserRequest(userName));
```

```
  // Attach an existing managed policy
  await iamClient.AttachUserPolicyAsync(new AttachUserPolicyRequest{
    UserName = responseCreate.User.UserName,
    PolicyArn = policyArn});

  // Create credentials and write them to a CSV file.
  CreateAccessKeyResponse responseCreds =
    await iamClient.CreateAccessKeyAsync(new CreateAccessKeyRequest{
      UserName = responseCreate.User.UserName});
  using (FileStream s = new FileStream(csvFilename, FileMode.Create))
  using (StreamWriter writer = new StreamWriter(s))
  {
    writer.WriteLine("User name,Access key ID,Secret access key");
    writer.WriteLine("{0},{1},{2}", responseCreds.AccessKey.UserName,
      responseCreds.AccessKey.AccessKeyId,
      responseCreds.AccessKey.SecretAccessKey);
  }

  return responseCreate;
}


//
// Method to print out a list of the existing users and information about them
private static async Task ListUsers(IAmazonIdentityManagementService iamClient)
{
  // Get the list of users
  ListUsersResponse responseUsers = await iamClient.ListUsersAsync();
  Console.WriteLine("\nFull list of users...");
  foreach (var user in responseUsers.Users)
  {
    Console.WriteLine($"User {user.UserName}:");
    Console.WriteLine($"\tCreated: {user.CreateDate.ToShortDateString()}");

    // Show the list of groups this user is part of
    ListGroupsForUserResponse responseGroups =
      await iamClient.ListGroupsForUserAsync(
        new ListGroupsForUserRequest(user.UserName));
    foreach (var group in responseGroups.Groups)
      Console.WriteLine($"\tGroup: {group.GroupName}");

    // Show the list of access keys for this user
    ListAccessKeysResponse responseAccessKeys =
      await iamClient.ListAccessKeysAsync(
        new ListAccessKeysRequest{UserName = user.UserName});
    foreach(AccessKeyMetadata accessKey in responseAccessKeys.AccessKeyMetadata)
      Console.WriteLine($"\tAccess key ID: {accessKey.AccessKeyId}");

    // Show the list of managed policies attached to this user
    var requestManagedPolicies = new ListAttachedUserPoliciesRequest{
      UserName = user.UserName};
    ListAttachedUserPoliciesResponse responseManagedPolicies =
      await iamClient.ListAttachedUserPoliciesAsync(
        new ListAttachedUserPoliciesRequest{UserName = user.UserName});
    foreach(var policy in responseManagedPolicies.AttachedPolicies)
      Console.WriteLine($"\tManaged policy name: {policy.PolicyName}");

    // Show the list of inline policies attached to this user
    ListUserPoliciesResponse responseInlinePolicies =
      await iamClient.ListUserPoliciesAsync(
        new ListUserPoliciesRequest(user.UserName));
    foreach(var policy in responseInlinePolicies.PolicyNames)
      Console.WriteLine($"\tInline policy name: {policy}");
  }
}
```

```
    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: IamCreateUser -u <user-name> -p <policy-arn> -c <csv-filename>" +
        "\n  -u, --user-name: The name of the user you want to create." +
        "\n  -p, --policy-arn: The ARN of an existing managed policy." +
        "\n  -c, --csv-filename: The name of a .csv file to write the credentials 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);
    }
  }

}
```

追加の考慮事項

- ユーザーのリストと、この例の結果については、IAM コンソール。

IAM ユーザーの削除

この例では、AWS SDK for .NETIAM ユーザーを削除します。まず、アクセスキー、アタッチされたポリシーなどのリソースを削除してから、ユーザーを削除します。

ここで示している各セクションで、この例のスニペットを学習できます。-例の完全なコード (p. 141)を後に示し、そのままビルドして実行できる。

トピック

- ユーザーからアイテムを削除する (p. 140)
- ユーザーを削除します。 (p. 141)
- コードの完了 (p. 141)
- 追加の考慮事項 (p. 144)

ユーザーからアイテムを削除する

次のスニペットは、ユーザーを削除する前にユーザーから削除する必要がある項目、管理ポリシーやアクセスキーなどの項目の例を示しています。

の例このトピックの最後にある (p. 141)に、このスニペットが使用中であることを示します。

```
    //
    // Method to detach managed policies from a user
    private static async Task DetachPolicies(
      IAmazonIdentityManagementService iamClient, string userName)
    {
      ListAttachedUserPoliciesResponse responseManagedPolicies =
        await iamClient.ListAttachedUserPoliciesAsync(
```

```
        new ListAttachedUserPoliciesRequest{UserName = userName});
    foreach(AttachedPolicyType policy in responseManagedPolicies.AttachedPolicies)
    {
      Console.WriteLine($"\tDetaching policy {policy.PolicyName}");
      await iamClient.DetachUserPolicyAsync(new DetachUserPolicyRequest{
        PolicyArn = policy.PolicyArn,
        UserName = userName});
    }
  }


  //
  // Method to delete access keys from a user
  private static async Task DeleteAccessKeys(
    IAmazonIdentityManagementService iamClient, string userName)
  {
    ListAccessKeysResponse responseAccessKeys =
      await iamClient.ListAccessKeysAsync(
        new ListAccessKeysRequest{UserName = userName});
    foreach(AccessKeyMetadata accessKey in responseAccessKeys.AccessKeyMetadata)
    {
      Console.WriteLine($"\tDeleting Access key {accessKey.AccessKeyId}");
      await iamClient.DeleteAccessKeyAsync(new DeleteAccessKeyRequest{
        UserName = userName,
        AccessKeyId = accessKey.AccessKeyId});
    }
  }
```

ユーザーを削除します。

次のスニペットは、メソッドを呼び出してユーザーから項目を削除し、そのユーザーを削除します。

の例このトピックの最後にある (p. 141)に、このスニペットが使用中であることを示します。

```
  //
  // Method to delete a user
  private static async Task DeleteUser(
    IAmazonIdentityManagementService iamClient, string userName)
  {
    Console.WriteLine($"\nDeleting user {userName}...");
    //
    // Remove items from the user
    //
    // Detach any managed policies
    await DetachPolicies(iamClient, userName);

    // Delete any access keys
    await DeleteAccessKeys(iamClient, userName);

    // DeleteLoginProfileAsycn(), DeleteUserPolicyAsync(), etc.
    // See the description of DeleteUserAsync for a full list.

    //
    // Delete the user
    //
    await iamClient.DeleteUserAsync(new DeleteUserRequest(userName));
    Console.WriteLine("Done");
  }
```

コードの完了

このセクションでは、関連する参考資料と、この例の完全なコードを示します。

SDK リファレンス

NuGet パッケージ:

- awssdk.IDENTITY 管理

プログラミング要素:

- 名前空間Amazon.IDENTITY 管理

  クラスAmazonIdentityManagementServiceClient
- 名前空間Amazon.identityManagement.model

  クラスアクセスキーメタデータ

  クラスアタッチされたポリシータイプ

  クラスDeleteAccessKeyRequest

  クラスDeleteUserRequest

  クラスユーザーポリシーリクエストのデタッチ

  クラスListAccessKeysRequest

  クラスListAccessKeysResponse

  クラスリストされたユーザーポリシーのリクエスト

  クラスリストされたユーザーポリシーレスポンス

コード

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

namespace IamDeleteUser
{
  class Program
  {
    static async Task Main(string[] args)
    {
      if(args.Length != 1)
      {
        Console.WriteLine("\nUsage: IamDeleteUser user-name");
        Console.WriteLine("   user-name - The name of the user you want to delete.");
        return;
      }

      // Create an IAM service client
      var iamClient = new AmazonIdentityManagementServiceClient();

      // Delete the given user
      await DeleteUser(iamClient, args[0]);

      // Could display a list of the users that are left.
    }

    //
```

```
    // Method to delete a user
    private static async Task DeleteUser(
      IAmazonIdentityManagementService iamClient, string userName)
    {
      Console.WriteLine($"\nDeleting user {userName}...");
      //
      // Remove items from the user
      //
      // Detach any managed policies
      await DetachPolicies(iamClient, userName);

      // Delete any access keys
      await DeleteAccessKeys(iamClient, userName);

      // DeleteLoginProfileAsycn(), DeleteUserPolicyAsync(), etc.
      // See the description of DeleteUserAsync for a full list.

      //
      // Delete the user
      //
      await iamClient.DeleteUserAsync(new DeleteUserRequest(userName));
      Console.WriteLine("Done");
    }


    //
    // Method to detach managed policies from a user
    private static async Task DetachPolicies(
      IAmazonIdentityManagementService iamClient, string userName)
    {
      ListAttachedUserPoliciesResponse responseManagedPolicies =
        await iamClient.ListAttachedUserPoliciesAsync(
          new ListAttachedUserPoliciesRequest{UserName = userName});
      foreach(AttachedPolicyType policy in responseManagedPolicies.AttachedPolicies)
      {
        Console.WriteLine($"\tDetaching policy {policy.PolicyName}");
        await iamClient.DetachUserPolicyAsync(new DetachUserPolicyRequest{
          PolicyArn = policy.PolicyArn,
          UserName = userName});
      }
    }


    //
    // Method to delete access keys from a user
    private static async Task DeleteAccessKeys(
      IAmazonIdentityManagementService iamClient, string userName)
    {
      ListAccessKeysResponse responseAccessKeys =
        await iamClient.ListAccessKeysAsync(
          new ListAccessKeysRequest{UserName = userName});
      foreach(AccessKeyMetadata accessKey in responseAccessKeys.AccessKeyMetadata)
      {
        Console.WriteLine($"\tDeleting Access key {accessKey.AccessKeyId}");
        await iamClient.DeleteAccessKeyAsync(new DeleteAccessKeyRequest{
          UserName = userName,
          AccessKeyId = accessKey.AccessKeyId});
      }
    }
  }
}
```

追加の考慮事項

- ユーザーから削除する必要があるリソースの詳細については、を参照してくださ
い。DeleteUserAsyncメソッドですが、参照されるメソッドの非同期バージョンを必ず使用してくださ
い。

- ユーザーのリストと、この例の結果については、IAM コンソール。

JSON から IAM 管理ポリシーを作成する

この例では、AWS SDK for .NET作成するにはIAM 管理ポリシーJSON の特定のポリシードキュメントか
ら。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
読み取り、ポリシーを作成します。

> Note
>
> JSON のポリシードキュメントの例については、を参照してください。追加の考慮事
> 項 (p. 147)このトピックの最後にあります。

ここで示している各セクションで、この例のスニペットを学習できます。-例の完全なコード (p. 144)を
後に示し、そのままビルドして実行できる。

トピック

- ポリシーを作成する (p. 144)
- コードの完了 (p. 144)
- 追加の考慮事項 (p. 147)

ポリシーを作成する

次のスニペットは、指定された名前とポリシードキュメントを使用して IAM 管理ポリシーを作成します。

の例このトピックの最後にある (p. 144)に、このスニペットが使用中であることを示します。

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

コードの完了

このセクションでは、関連する参考資料と、この例の完全なコードを示します。

SDK リファレンス

NuGet パッケージ:

- awssdk.IDENTITY 管理

プログラミング要素:

- 名前空間Amazon.IDENTITY 管理

  クラスAmazonIdentityManagementServiceClient
- 名前空間Amazon.identityManagement.model

  クラスCreatePolicyRequest

  クラスCreatePolicyResponse

コード

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Amazon.IdentityManagement;
using Amazon.IdentityManagement.Model;

namespace IamCreatePolicyFromJson
{
  // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
 =
  // 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、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るポリシードキュメントの例を示します。

```
{
  "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確認するには、IAM コンソール。左ポリシーのフィルタリングドロップダウンリストから [] を選択します。カスタマー管理型。不要になったポリシーを削除します。

- ポリシー作成の詳細については、「」を参照してください。IAM ポリシーの作成とIAM JSON ポリシーリファレンスのIAM ユーザーガイド

IAM 管理ポリシーのポリシードキュメントを表示する

この例では、AWS SDK for .NETをクリックして、ポリシードキュメントを表示します。アプリケーションによって IAM クライアントオブジェクトが作成され、指定された IAM 管理ポリシーのデフォルトバージョンが検出され、ポリシードキュメントが JSON で表示されます。

ここで示している各セクションで、この例のスニペットを学習できます。-例の完全なコード (p. 149)を後に示し、そのままビルドして実行できる。

トピック

- デフォルトバージョンを検索する (p. 148)
- ポリシードキュメントの表示 (p. 148)
- コードの完了 (p. 149)

デフォルトバージョンを検索する

次のスニペットは、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。

の例このトピックの最後にある (p. 149)に、このスニペットが使用中であることを示します。

```
//
// 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

次のスニペットは、指定された IAM ポリシーの JSON でポリシードキュメントを表示します。

の例このトピックの最後にある (p. 149)に、このスニペットが使用中であることを示します。

```
//
```

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

コードの完了

このセクションでは、関連する参考資料と、この例の完全なコードを示します。

SDK リファレンス

NuGet パッケージ:

- awssdk.IDENTITY 管理

プログラミング要素:

- 名前空間Amazon.IDENTITY 管理

  クラスAmazonIdentityManagementServiceClient
- 名前空間Amazon.identityManagement.model

  クラスGetPolicyVersionリクエスト

  クラスGetPolicyVersion レスポンス

  クラスListPolicyVersions リクエスト

  クラスListPolicyVersionsレスポンス

  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 ロールを使用してアクセスを許可する

このチュートリアルでは、の使い方を学習できます。AWS SDK for .NETをクリックして、Amazon EC2 インスタンスで IAM ロールを有効にします。

Overview

へのすべてのリクエストAWSが発行した認証情報を使用して暗号で署名される必要があります。AWS。したがって、Amazon EC2 インスタンスで実行されるアプリケーション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安全に配信、保存、およびローテーションする必要がありますが、アプリケーションにアクセスできるようにする必要があります。

IAM ロールを使用すると、これらの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権限を使用して設定し、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、「」を参照してください。Linux 用 EC2 ユーザーガイドまたはWindows 用 EC2 ユーザーガイド。また、次の情報も参照してください。IAM ロールIAM ユーザーガイド

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、クライアントオブジェクトをAWSサービスでは、オブジェクトはいくつかの潜在的なソースから資格情報を検索します。は、検索順序を示します。認証情報とプロファイルの解決 (p. 25)。

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ることができなかった場合、IAM ロールに設定され、EC2 インスタンスのメタデータ内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。これらのクレデンシャルは、を呼び出すために使用されます。AWSクライアントオブジェクトから。

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、AWS SDK for .NET（およびその他のツール）、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 ロールのアクセス権限を使用してインスタンス上のアプリケーションを表示します。

トピック

サンプル Amazon S3 アプリケーションを作成する

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、以下のものが必要です。

- 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、「」をご参照ください。Amazon S3 入門ガイド。についての情報AWSクレデンシャル、を参照してください。設定AWS認証情報 (p. 20)。

次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、.NET Core Runtime がインストールされ、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ように選択できます。これにより、同様のエクスペリエンスとファイル転送が小さくなります。
ただし、インスタンスに .NET Core Runtime をインストールしないことを選択することもできます。このアクションコースを選択した場合は、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、開発マシンで使用す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復元してください。)

IAM ロールを作成する

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 [] を選択します。ロール[] を選択してから、ロールの作成。
3. SelectAWSのサービス[] を検索して選択します。EC2を選択し、次へ: アクセス許可.

4. []アクセス許可ポリシーのアタッチ[] を検索して選択します。AmazonS3ReadOnlyAccess。必要に応じてポリシーを確認し、[] を選択します。次へ: タグ
5. 必要に応じてタグを追加し、次へ: 確認.
6. ロールの名前と説明を入力し、[Create role] を選択します。EC2 インスタンスを起動するときに必要になるため、この名前を忘れないでください。

EC2 インスタンスを起動し、IAM ロールをアタッチする

前に作成した 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 コンソールを使用する

  指示に従って、でインスタンスを起動します。Linux 用 EC2 ユーザーガイドまたはWindows 用 EC2 ユーザーガイド。

  ウィザードを進めると、少なくとも、インスタンスの詳細の設定ページを選択して、IAM ロール先ほど作成したこと。

- AWS SDK for .NET の使用

  詳細については、「」を参照してください。Amazon EC2 インスタンスの起動 (p. 113)を含む。追加の考慮事項 (p. 120)そのトピックの最後近くに。

IAM ロールがアタッチされた 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、を参照してください。Linux 用 EC2 ユーザーガイドまたはWindows 用 EC2 ユーザーガイド。

EC2 インスタンスConnect。

EC2 インスタンスConnect して、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、アプリケーションを実行できるようにします。インスタンスの起動に使用したkey pair プライベート部分、つまり PEM ファイルが含まれているファイルが必要です。

これを行うには、の接続手順に従います。Linux 用 EC2 ユーザーガイドまたはWindows 用 EC2 ユーザーガイド。接続するときは、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、「」を参照してください。Amazon EC2 インスタンスへの接続の AWS Toolkit for Visual Studioユーザーガイド。

EC2 インスタンスでサンプルアプリケーションを実行します。

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。インスタンスへのファイルの転送方法については、を参照してください。Linux 用 EC2 ユーザーガイドまたはWindows 用 EC2 ユーザーガイド。
2. アプリケーションを起動し、開発マシンと同じ結果で実行されていることを確認します。
3. アプリケーションが IAM ロールによって提供される認証情報を使用し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、IAM ロールをデタッチします。アクション,インスタンス設定,IAM ロールのアタッチ/置き換え。
   c. アプリケーションを再度実行して、認証エラーが返されることを確認します。

クリーンアップ

このチュートリアルを終了し、作成した EC2 インスタンスが不要になった場合は、不要なコストを避けるためにインスタンスを終了してください。[] でこれを行うことができます。Amazon EC2 コンソールまたはプログラム的に、で説明するようにAmazon EC2 インスタンスの削除 (p. 124)。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を削除することもできます。これには、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含まれます。

Amazon Simple Storage Service インターネットストレージの使用

-AWS SDK for .NETが をサポートAmazon S3がインターネット用のストレージサービスです。また、ウェブスケールのコンピューティングを開発者が簡単に利用できるよう設計されています。

APIs

-AWS SDK for .NETでは、Amazon S3 クライアント用の API を提供します。API を使用すると、バケットやアイテムなどの Amazon S3 リソースを操作できます。Amazon S3 の 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 .NETAPI リファレンス(「Amazon.S3」までスクロールします)。
- Amazon.Extensions.S3.Encryptionドキュメント

Amazon S3 API は、次の NuGet パッケージによって提供されます。

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

Prerequisites

開始する前に、必ず次のものを確認してください。環境をセットアップする (p. 15)。また、「」の情報を確認します。プロジェクトをセットアップする (p. 17)およびSDK の機能 (p. 49)。

このドキュメントの例

このドキュメントの次のトピックでは、の使用方法を示します。AWS SDK for .NETAmazon S3 での作業を行う。

- S3 暗号化に KMS キーを使用する (p. 157)

他のドキュメントの例

次のリンクはAmazon S3 開発者ガイドの使用方法の追加の例を示します。AWS SDK for .NETAmazon S3 での作業を行う。

> **Note**
>
> これらの例と追加のプログラミングに関する考慮事項は、のバージョン 3 用に作成されていますがAWS SDK for .NET.NET Framework を使用すると、それ以降のバージョンのAWS SDK for .NET.NET Core を使用する。コード内の小さな調整が必要な場合があります。

Amazon S3 の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.transfer.Transfer ユーティリティ)
- 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

- の使用AWS SDK for .NETAmazon S3 プログラミングの場合
- 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

を使用するAWS KMSでの Amazon S3 暗号化のキー AWS SDK for .NET

この例では、の使用方法を示します。AWS Key Management ServiceAmazon S3 オブジェクトを暗号化するためのキー。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Amazons3EncryptionClientV2クライアント側の暗号化のオブジェクト。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 バケット内の特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し、その内容を表示します。

> Warning
>
> という類似のクラスAmazonS3EncryptionClientは非推奨であり、より安全性が低いAmazonS3EncryptionClientV2クラス。を使用する既存のコードを移行するにはAmazonS3EncryptionClient「」を参照してください。S3 暗号化クライアントの移行 (p. 211)。

トピック

- 暗号化マテリアルを作成する (p. 158)

暗号化マテリアルを作成する

次のスニペットでは、`EncryptionMaterials`KMS キー ID を含むオブジェクト。

の例このトピックの最後にある (p. 158)に、このスニペットが使用中であることを示します。

```
     // 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 オブジェクトを作成して暗号化する

次のスニペットでは、`AmazonS3EncryptionClientV2`以前に作成した暗号化マテリアルを使用するオブジェクト。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て暗号化します。

の例このトピックの最後にある (p. 158)に、このスニペットが使用中であることを示します。

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

コードを完成させる

このセクションでは、この例の関連するリファレンスと完全なコードを示します。

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

  クラス暗号化マテリアル V2
- 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.keyManagementサービス

  クラスAmazonKey管理サービスクライアント
- 名前空間Amazon.keyManagementService.model

  クラスCreateKey リクエストの作成

  クラスキーレスポンスの作成

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。これを行うには、「」を参照してください。Amazon S3 コンソールアプリケーションに提供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、標準と同じインターフェイスを実装しています。AmazonS3Clientクラス。これにより、コードを移植しやすくなります。AmazonS3EncryptionClientV2暗号化と復号化がクライアントで自動的かつ透過的に行われるようにするクラス。

- を使用する利点の 1 つAWS KMSキーをマスターキーとして使用する場合は、独自のマスターキーを保存および管理する必要がないということです。AWS。2つ目の利点は、AmazonS3EncryptionClientV2のクラスAWS SDK for .NETとの相互運用が可能であるAmazonS3EncryptionClientV2のクラスAWS SDK for Java。つまり、AWS SDK for Java で暗号化して AWS SDK for .NET で復号でき、その逆も可能です。

  Note

  -AmazonS3EncryptionClientV2のクラスAWS SDK for .NETでは、メタデータモードで実行している場合にのみ、KMS マスターキーがサポートされています。の命令ファイルモードAmazonS3EncryptionClientV2のクラスAWS SDK for .NETとは互換性がありませんAmazonS3EncryptionClientV2のクラスAWS SDK for Java。

- を使用したクライアント側の暗号化の詳細については、を参照してくださいAmazonS3EncryptionClientV2クラス、およびエンベロープ暗号化の仕組みについては、を参照してください。を使用したクライアント側のデータ暗号化AWS SDK for .NETおよび Amazon S3 におけるもの。

Amazon Simple Notification サービスを使用したクラウドからの通知の送信

Note

このトピックの情報は、.NET Framework およびAWS SDK for .NETバージョン 3.3 以降

-AWS SDK for .NETは、アプリケーション、エンドユーザー、およびデバイスでクラウドからすぐに通知を送信できるウェブサービスである Amazon Simple Notification Service (Amazon SNS) をサポートします。詳細については、「Amazon SNS」を参照してください。

Amazon SNS トピックの出品

次の例は、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 AmazonSimpleNotificationServiceClient を使用します。これは、デフォルト設定から認証情報をロードします。

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

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

do
{
  response = client.ListTopics(request);

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

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

    var ss = subs.Subscriptions;

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

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

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

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

      foreach (var attr in attrs)
```

```
      {
        Console.WriteLine("    {0} = {1}", attr.Key, attr.Value);
      }
    }

    Console.WriteLine();
  }

  request.NextToken = response.NextToken;

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

Amazon SNS トピックへのメッセージの発行

次の例は、Amazon SNS トピックにメッセージを送信する方法を示します。この例では、1 つの引数 (Amazon SNS トピックの ARN) を取ります。

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

using Amazon;
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;

namespace SnsSendMessage
{
    class Program
    {
        static void Main(string[] args)
        {
            /* Topic ARNs must be in the correct format:
             *   arn:aws:sns:REGION:ACCOUNT_ID:NAME
             *
             *  where:
             *  REGION     is the region in which the topic is created, such as us-west-2
             *  ACCOUNT_ID is your (typically) 12-character account ID
             *  NAME       is the name of the topic
             */
            string topicArn = args[0];
            string message = "Hello at " + DateTime.Now.ToShortTimeString();

            var client = new AmazonSimpleNotificationServiceClient(region:
 Amazon.RegionEndpoint.USWest2);

            var request = new PublishRequest
            {
                Message = message,
                TopicArn = topicArn
            };

            try
            {
                var response = client.Publish(request);

                Console.WriteLine("Message sent to topic:");
                Console.WriteLine(message);
            }
            catch (Exception ex)
            {
                Console.WriteLine("Caught exception publishing request:");
                Console.WriteLine(ex.Message);
            }
```

```
        }
    }
}
```

GitHub で、コマンドラインからサンプルをビルドして実行する方法に関する情報を含む、完全な例を参照してください。

1 つの電話番号に SMS メッセージを送信する

次の例は、電話番号に SMS メッセージを送信する方法を示します。この例では、1 つの引数 (電話番号) を取ります。この引数は、コメントに記載されている 2 つの形式のいずれかである必要があります。

```
using System;
using System.Linq;
using System.Threading.Tasks;
using Amazon;
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;

namespace SnsPublish
{
    class Program
    {
        static void Main(string[] args)
        {
            // US phone numbers must be in the correct format:
            // +1 (nnn) nnn-nnnn OR +1nnnnnnnnnn
            string number = args[0];
            string message = "Hello at " + DateTime.Now.ToShortTimeString();

            var client = new AmazonSimpleNotificationServiceClient(region:
 Amazon.RegionEndpoint.USWest2);
            var request = new PublishRequest
            {
                Message = message,
                PhoneNumber = number
            };

            try
            {
                var response = client.Publish(request);

                Console.WriteLine("Message sent to " + number + ":");
                Console.WriteLine(message);
            }
            catch (Exception ex)
            {
                Console.WriteLine("Caught exception publishing request:");
                Console.WriteLine(ex.Message);
            }
        }
    }
}
```

GitHub で、コマンドラインからサンプルをビルドして実行する方法に関する情報を含む、完全な例を参照してください。

Amazon SQS を使用したメッセージング

-AWS SDK for .NETが をサポートAmazon Simple Queue Service (Amazon SQS)。これは、システム内の他のコンポーネントとの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、受信できるメカニズムを提供します。これにより、このようなコンポーネントを切り離し、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から解放されます。Amazon SQS でのキューとメッセージの仕組みについては、「」を参照してください。Amazon SQS チュートリアルおよびAmazon SQS の仕組みのAmazon Simple Queue Service 開発者ガイド。

> Important
>
> キューの分散性が原因となり、Amazon SQS では、メッセージが送信されたときに受信できるかどうかは保証されません。メッセージの順序を保持する必要がある場合は、Amazon SQS キュー。

APIs

-AWS SDK for .NETは、Amazon SQS クライアント用の API を提供します。API を使用すると、キューやメッセージなどの Amazon SQS 機能を操作できます。このセクションでは、これらの API を操作するときに従うことができるパターンを示す少数の例を紹介します。API の完全なセットを表示するには、を参照してください。AWS SDK for .NETAPI リファレンス(そして「Amazon.sqs」までスクロールします)。

Amazon SQS API は、AWSSDK.SQSNuGet パッケージです。

Prerequisites

始める前に、必ず次のものがあるかどうかを確認してください。環境をセットアップする (p. 15)。また、「」の情報を確認します。プロジェクトをセットアップする (p. 17)およびSDK の機能 (p. 49)。

Topics

トピック

- Amazon SQS キューを作成する (p. 166)
- Amazon SQS キューの更新 (p. 173)
- Amazon SQS キューを削除する (p. 178)
- Amazon SQS メッセージを送信する (p. 181)
- Amazon SQS メッセージの受信 (p. 185)

Amazon SQS キューを作成する

この例では、使用方法を示します。AWS SDK for .NETを使用して、Amazon SQS キューを作成する。このアプリケーションはを作成します。デッドレターキューARN を 1 つに供給しない場合。次に、デッドレターキュー (指定したキューまたは作成されたキュー) 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指定しない場合、アプリケーションは既存のすべてのキューに関する情報を表示するだけです。

次のセクションでは、この例のスニペットを学習できます。-例のための完全なコード (p. 168)を後に示し、そのままビルドして実行できる。

トピック

- 既存のキューを表示する (p. 167)
- キューを作成する (p. 167)
- キューの ARN を取得する (p. 168)
- コードの完了 (p. 168)
- 追加の考慮事項 (p. 172)

既存のキューを表示する

次のスニペットは、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、各キューの属性を示しています。

の例このトピックの最後にある (p. 168)このスニペットが使用中であることを示します。

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

キューを作成する

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。

の例このトピックの最後にある (p. 168)このスニペットが使用中であることを示します。

```
//
// 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 を取得する

次のスニペットは、指定されたキュー URL で識別されるキューの ARN を取得します。

の例このトピックの最後にある (p. 168)このスニペットが使用中であることを示します。

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

コードの完了

このセクションでは、この例の関連する参考文献と完全なコードを示します。

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GetQueueAttributes レスポン

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 が必要で、キュー名しかない場合は、`AmazonSQSClient.GetQueueUrlAsync`メソッド。

- 設定できるさまざまなキュー属性の詳細については、「」を参照してくださ
い。CreateQueueRequestのAWS SDK for .NETAPI リファレンスまたはSetQueueAttributesのAmazon Simple Queue Service API 参照。

- この例では、作成するキュー上のすべてのメッセージのロングポーリングを指定します。これは、`ReceiveMessageWaitTimeSeconds`属性。

  へのコール中にロングポーリングを指定することもできます。`ReceiveMessageAsync`の方法AmazonSQSClientクラス。詳細については、「Amazon SQS メッセージの受信 (p. 185)」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詳細については、「」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Amazon Simple Queue Service 開発者ガイド。

- デッドレターキューは、正常に処理できないメッセージの送信先として他の (送信元) キューが使用できるキューです。詳細については、「」を参照してください。Amazon SQS デッドレターキューAmazon Simple Queue Service 開発者ガイドを参照してください。

- また、キューのリストとこの例の結果をAmazon SQS コンソール。

Amazon SQS キューの更新

この例では、使用方法を示します。AWS SDK for .NETを使用して、Amazon SQS キューを更新できます。いくつかのチェックの後、アプリケーションは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 のみが含まれる場合、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。

次のセクションでは、この例のスニペットを学習できます。-例のための完全なコード (p. 174)を後に示し、そのままビルドして実行できる。

トピック

- キュー属性の表示 (p. 173)
- 属性名を検証する (p. 174)
- 更新キュー属性 (p. 174)
- コードの完了 (p. 174)
- 追加の考慮事項 (p. 178)

キュー属性の表示

次のスニペットは、指定されたキュー URL で識別されるキューの属性を示しています。

の例このトピックの最後にある (p. 174)このスニペットが使用中であることを示します。

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

属性名を検証する

次のスニペットは、更新される属性の名前を検証します。

の例このトピックの最後にある (p. 174)このスニペットが使用中であることを示します。

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

更新キュー属性

次のスニペットは、指定されたキュー URL で識別されるキューの属性を更新します。

の例このトピックの最後にある (p. 174)このスニペットが使用中であることを示します。

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

コードの完了

このセクションでは、この例の関連する参考文献と完全なコードを示します。

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GetQueueAttributes レスポン

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

- を更新するにはRedrivePolicy属性を使用する場合は、オペレーティングシステムに応じて、値全体を引用し、キーと値のペアの引用符を工スケープする必要があります。

  たとえば、Windows では、値は次のような方法で構築されます。

```
"{\"deadLetterTargetArn\":\"DEAD_LETTER-QUEUE-ARN\",\"maxReceiveCount\":\"10\"}"
```

Amazon SQS キューを削除する

この例では、使用方法を示します。AWS SDK for .NETを選択して、Amazon SQS キューを削除します。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、残りのキューのリストを表示します。

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既存のキューのリストを表示します。

次のセクションでは、この例のス二ペットを学習できます。-例のための完全なコード (p. 179)を後に示し、そのままビルドして実行できる。

トピック

キューの削除

次のス二ペットは、指定されたキュー URL で識別されるキューを削除します。

の例このトピックの最後にある (p. 179)このス二ペットが使用中であることを示します。

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

キューがなくなるのを待つ

次のス二ペットは、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。

の例このトピックの最後にある (p. 179)このス二ペットが使用中であることを示します。

```
    //
    // 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を表示する

次のスニペットは、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示しています。

の例このトピックの最後にある (p. 179)このスニペットが使用中であることを示します。

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

コードの完了

この節では、この例の関連する参考文献と完全なコードを次に示します。

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

- -DeleteQueueAsyncAPI 呼び出しでは、削除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。より洗練された手順でこれを確認できます。

- また、キューのリストとこの例の結果をAmazon SQS コンソール。

Amazon SQS メッセージを送信する

この例では、使用方法を示します。AWS SDK for .NET作成できる Amazon SQS キューにメッセージを送信するにはプログラムによる (p. 166)または、Amazon SQS コンソール。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入力を待機します。これは、キューに送信する追加のメッセージや、アプリケーションの終了要求などです。

この例と次の例：メッセージの受信について (p. 185)を一緒に使用して Amazon SQS でメッセージフローを確認できます。

次のセクションでは、この例のスニペットを学習できます。-例のための完全なコード (p. 183)を後に示し、そのままビルドして実行できる。

トピック

メッセージの送信

次のスニペットは、指定されたキュー URL で識別されるキューにメッセージを送信します。

の例このトピックの最後にある (p. 183)このスニペットが使用中であることを示します。

```
//
// 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を送信する

次のスニペットは、指定されたキュー URL で識別されるキューにメッセージのバッチを送信します。

の例このトピックの最後にある (p. 183)このスニペットが使用中であることを示します。

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

キューからすべてのメッセージを削除する

次のスニペットは、指定されたキュー URL で識別されるキューからすべてのメッセージを削除します。これは、キューの消去。

の例このトピックの最後にある (p. 183)このスニペットが使用中であることを示します。

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

コードの完了

この節では、この例の関連する参考文献と完全なコードを次に示します。

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

  クラスパージキューレスポンス

  クラスSendMessageBatchResponse

  クラスSendMessageResponse

  クラスSendMessageBatchRequestEntry

  クラスSendMessageBatch結果エントリ

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、を参照してください。メッセージに関連するクォータのAmazon Simple Queue Service 開発者ガイド。

- メッセージは削除されるか、キューがパージされるまで、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、キュー内にまだ存在していても、そのメッセージはキューに表示されません。可視性タイムアウトの詳細については、「」を参照してください。Amazon SQS 可視性タイムアウト。

- 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、「」を参照してください。メッセージメタデータ。

Amazon SQS メッセージの受信

この例では、使用方法を示します。AWS SDK for .NET作成できる Amazon SQS キューからメッセージを受信するにはプログラムによる (p. 166)または、Amazon SQS コンソール。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、メッセージを処理し (この場合は、コンソールにメッセージ本文を表示します) 、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にキーを入力するまで、これらの手順を繰り返します。

この例とメッセージの送信に関する前の例 (p. 181)を一緒に使用して Amazon SQS でメッセージフローを確認できます。

次のセクションでは、この例のスニペットを学習できます。-例のための完全なコード (p. 186)を後に示し、そのままビルドして実行できる。

トピック

- メッセージを受信する (p. 186)

メッセージを受信する

次のスニペットは、指定されたキュー URL で識別されるキューからメッセージを受け取ります。

の例このトピックの最後にある (p. 186)このスニペットが使用中であることを示します。

```
//
// 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

次のスニペットは、指定されたキュー URL で識別されるキューからメッセージを削除します。

の例このトピックの最後にある (p. 186)このスニペットが使用中であることを示します。

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

コードの完了

このセクションでは、この例の関連する参考文献と完全なコードを示します。

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 る

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には、この例ではWaitTimeSecondsへの各呼び出しのプロパティReceiveMessageAsyncメソッド。

  また、を使用して、キュー上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。ReceiveMessageWaitTimeSeconds属性時作成中 (p. 166)または更新 (p. 173)キュー。

  ショートポーリングとロングポーリングの詳細については、「」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Amazon Simple Queue Service 開発者ガイド。

- 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。これを行う方法については、「」を参照してください。ChangeMessageVisibilityAsyncの方法AmazonSQSClientクラス。

- を呼び出します。DeleteMessageAsyncメソッドは、可視性のタイムアウト設定にかかわらず、メッセージをキューから無条件で削除します。

プログラミングAWS OpsWorksするには、スタックとアプリケーションを処理する

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。とAWS OpsWorksでは、プロビジョニングできますAWSリソース、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。詳細については、「」を参照してください。OpsWorks 製品ページとAWS OpsWorksユーザーガイド。

APIs

-AWS SDK for .NETの API を提供します。AWS OpsWorks。API を使用すると、AWS OpsWorks次のような機能を備えています。スタック彼らとレイヤー,インスタンス, およびアプリケーション。API の完全なセットを表示するには、AWS SDK for .NETAPI リファレンス(そして「Amazon.opsWorks」までスクロールします)。

-AWS OpsWorksAPI は、awssdk.opsWorksNuGet パッケージ。

Prerequisites

開始する前に、必ず次の機能を備えています。使用する環境をセットアップする (p. 15)。また、「」の情報を確認します。プロジェクトをセットアップする (p. 17)およびSDK の機能 (p. 49)。

その他の SupportAWSのサービスと設定

-AWS SDK for .NETが をサポートAWSサービスに加えて、前のセクションで説明しているサービスもあります。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 .NETAPI リファレンス。

個々の名前空間に加えてAWSサービスでは、AWS SDK for .NETでは、次の API がサポートされています。

エリア	説明	リソース
AWS サポート	へのプログラムによるアクセスAWSSupport ケースおよび Trusted Advisor 機能。	「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。

とともに使用する追加ツールはAWS SDK for .NET

このセクションでは、の追加ツールについて説明します。追加ツールは、AWS SDK for .NET。これらのツールは、アプリケーションの開発、デプロイ、保守に役立ちます。

これらのツールは、環境をセットアップする (p. 15),プロジェクトをセットアップする (p. 17)と連動するアプリケーションの作成AWSサービス、その例についてはの使用AWSのサービス (p. 69)。

使用できるツールは次のとおりです。

- AWS.NET CLI用の.NET配置ツール (p. 190)

AWS.NET CLI用の.NET配置ツール

> 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、そのアプリケーションをAWS。

へのデプロイAWS時には、複数のAWSサービスおよびリソースで構成する必要があります。このデプロイメント作業を容易にするために、AWS.NET CLI用の.NET配置ツール、またはデプロイツール略して。

.NET Core アプリケーション用の配置ツールを実行すると、ツールによってすべてのAWSアプリケーションのデプロイに使用できるコンピュートサービスオプション。これは、最も可能性の高い選択肢と、その選択肢に沿って最も可能性の高い設定を示唆しています。次に、選択したコンピュートサービスで必要に応じてアプリケーションをビルドし、パッケージ化します。デプロイメントインフラストラクチャを生成し、AWS Cloud Development Kit (CDK)をクリックし、エンドポイントを表示します。

デプロイメントオプションを対話的に選択するか、JSON 設定ファイル (p. 200)。また、ツールによって自動的に選択されるデフォルト値を維持することもできます。

デプロイメントツールに関する追加情報は、GitHub レポ (https://github.com/aws/aws-dotnet-deployとブログの記事再イメージ化AWS.NET デプロイの経験。

Capabilities

- へのデプロイAWS Elastic Beanstalkまたは Amazon ECS (AWS Fargate).
- .NET Core 2.1 以降で構築され、Linux に展開する目的で作成されたクラウドネイティブの .NET アプリケーションをデプロイします。このようなアプリケーションは、Windows レジストリ、IIS、MSMQ などの Windows 固有のテクノロジには結び付けられず、仮想コンピューティングに展開できます。
- ASP.NET コア Web アプリケーション、Blazor WebAssembly アプリケーション、長時間実行されるサービスアプリケーション、およびスケジュールされたタスクを展開します。詳細については、「」を参照してください。READMEをGitHub リポジトリに追加します。

トピック

環境をセットアップする

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。

次のセクションで、デプロイツールを実行するための環境をセットアップする方法について説明します。

Node.js

デプロイメントツールには、AWS Cloud Development Kit (CDK)であり、AWS CDKには が必要ですNode.jsバージョン 10.13.0 以降 (バージョン 13.0.0 から 13.6.0 を除く)。インストールした Node.js のバージョンを確認するには、コマンドプロンプトまたはターミナルで次のコマンドを実行します。

```
node --version
```

Note

そのファイルにAWS CDKがマシンにインストールされていない場合、またはAWS CDKが最小バージョン (1.95.2) より前のバージョンである場合、デプロイメントツールはツールでのみ使用される CDK の一時的な「プライベート」コピーをインストールし、マシンのグローバル設定は変更されません。
代わりに、インストールするクライアントAWS CDKについては、のインストールAWS CDKのAWS Cloud Development Kit (CDK)デベロッパーガイド

.NET Core と .NET Core

アプリケーションは、.NET Core 3.1 以降から構築する必要があります (.NET Core 3.1、.NET 5.0 など)。使用しているバージョンを確認するには、コマンドプロンプトまたはターミナルで次のコマンドを実行します。

```
dotnet --version
```

.NET をインストールまたは更新する方法の詳細については、」https://dotnet.microsoft.com/。

（オプション）ドッカー

Amazon Elastic Container Service (Amazon ECS) にアプリケーションをデプロイする場合、AWS Fargate を使用する場合は、デプロイメントツールを実行する場所にDockerをインストールする必要があります。詳細については、「」を参照してください。https://docs.docker.com/engine/install/。

(Linux および macOS) ZIP CLI

ZIP CLI は、デプロイメントバンドル用の ZIP パッケージを作成するときに使用されます。これは、Linux のファイル権限を維持するために使用されます。

デプロイツールを設定する

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。

次の手順では、デプロイツールをインストール、更新、およびアンインストールする方法について説明します。

デプロイツールをインストールするには

1. コマンドプロントまたはターミナルを開きます。
2. ツールをインストールします。`dotnet tool install --global aws.deploy.cli`
3. バージョンを確認して、インストールを確認します。`dotnet aws --version`

デプロイツールを更新するには

1. コマンドプロントまたはターミナルを開きます。
2. バージョンを確認します。`dotnet aws --version`
3. (省略可能) 新しいバージョンのツールがデプロイツールの NuGet ページ。
4. ツールを更新します。`dotnet tool update -g aws.deploy.cl`インスタンス
5. バージョンを再度確認して、インストールを確認します。`dotnet aws --version`

システムから展開ツールを削除するには

1. コマンドプロントまたはターミナルを開きます。
2. ツールをアンインストールします。`dotnet tool uninstall -g aws.deploy.cli`
3. ツールがインストールされていないことを確認します。`dotnet aws --version`

デプロイツールの認証情報を設定します

これはプレビューリリースの機能に関するプレリリースドキュメントです。このドキュメントは変更される可能性があります。

ここでは、デプロイツールの認証情報を設定する方法について説明します。プロジェクトの認証情報を設定する方法の詳細については、」設定AWS認証情報 (p. 20)代わりに、します。

デプロイツールをに実行するにはAWSアカウントにログインするには、資格情報プロファイルが必要です。プロファイルは、少なくとも、アクセスキー ID とシークレットアクセスキーを使用して設定する必要があります。AWS Identity and Access Management(IAM) ユーザーです。これを行う方法については、」ユーザーとロールの作成 (p. 18)および共有AWS認証情報ファイル (p. 21)。

展開ツールの実行に使用する資格情報には、実行しようとしているタスクに応じて、特定のサービスに対するアクセス許可が必要です。次に、ツールの実行に必要な一般的なアクセス許可の例をいくつか示しま

す。デプロイするアプリケーションのタイプと使用するサービスによっては、追加の権限が必要になる場合があります。

タスク	サービスのアクセス権限
次のリストを表示します。AWS CloudFormationスタック (リストデプロイ)	CloudFormation
Elastic Beanstalk へのデプロイと再デプロイ (デプロイ)	CloudFormation、Elastic Beanstalk
Amazon ECS へのデプロイと再デプロイ (デプロイ)	CloudFormation、Elastic Beanstalk、弾性コンテナレジストリ

デプロイメントツールでは、`[default]`プロファイルから共有AWS設定ファイルおよび認証情報ファイル (p. 21)そのプロファイルが存在する場合。この動作を変更するには、ツールで使用するプロファイルを、システム全体または特定のコンテキストで指定します。

システム全体のプロファイルを指定するには、次の手順を実行します。

- [] を指定します。`AWS_PROFILE`オペレーティングシステムに応じて、環境変数をグローバルに実行します。必要に応じて、コマンドプロンプトまたは端子を再度開いてください。指定したプロファイルにAWS領域を選択すると、ツールによって選択を求められる場合があります。

  Warning

  設定した場合`AWS_PROFILE`環境変数をシステムにグローバルに適用すると、他の SDK、CLI、ツールもそのプロファイルを使用します。この動作が許容できない場合は、代わりに特定のコンテキストのプロファイルを指定します。

特定のコンテキストにプロファイルを指定するには、次のいずれかを実行します。

- [] を指定します。`AWS_PROFILE`環境変数を、ツールを実行しているコマンドプロンプトまたはターミナルセッションで使用します (オペレーティングシステムに応じて)。
- [] を指定します。`--profile`および`--region`スイッチをコマンド内で実行します。以下に例を示します。`dotnet aws list-stacks --region us-west-2`。 展開ツールのコマンドの詳細については、」デプロイツールの実行 (p. 193)。
- 何も指定しないと、ツールはプロファイルとAWSリージョン。

デプロイツールの実行

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。

展開ツールに慣れるために、このトピックでは、ツールのコマンドラインのクイックツアーについて説明します。

コマンドラインヘルプ

基本的なヘルプを表示するには、以下を実行します。

```
dotnet aws --help
```

使用可能なコマンドの一覧は、コマンドセクション (例:`deploy`および`list-deployments`。

特定のコマンドに関するヘルプを表示するには、`--help`そのコマンドのコンテキストで。以下に例を示します。

```
dotnet aws deploy --help
```

いくつかの一般的なコマンド

アプリケーションをデプロイまたは再デプロイする手順は、次のとおりです。

```
cd <dotnet_core_app_directory>
dotnet aws deploy [--profile <profile_name>] [--region <region_code>] [--apply
 <configuration_file>] [-s|--silent]
```

ツールを使用して作成した CloudFormation スタックのリストを表示するには、次の手順に従います。

```
dotnet aws list-deployments [–region <region_code>] [--profile <profile_name>]
```

アプリケーションをデプロイするためのチュートリアル

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。

アプリケーションを開発したら、AWS。展開ツールを使用すると、ツールによってすべてのAWSアプリケーションのデプロイに使用できるコンピュートサービスオプション。次に、最も可能性の高い選択肢と、その選択肢に沿った可能性の高い設定が示唆されます。

次のセクションで、デプロイの例を示します。AWS。例では初歩的なアプリケーション、`[default]`の資格情報、および展開ツールによって提供されるデフォルト設定が含まれます。

チュートリアル

- Elastic Beanstalk へのウェブアプリケーションのデプロイ (p. 194)
- Amazon ECS へのウェブアプリケーションのデプロイ (p. 196)

Elastic Beanstalk へのウェブアプリケーションのデプロイ

これはプレビューリリースの機能に関するプレリリースドキュメントです。このドキュメントは変更される可能性があります。

このチュートリアルでは、Elastic Beanstalk へのデプロイについて説明します。このチュートリアルでは、配置ツールによって提供されるデフォルトを使用します。

Prerequisites

- これで、完了です。環境のセットアップ (p. 191)およびツールのセットアップ (p. 192)。
- `-[default]`認証情報プロファイルには、必要なアクセス許可があります。

Deploy

アプリケーションをデプロイする準備ができたらAWS初めて、これが始める場所です。

サンプル Web アプリケーションを Elastic Beanstalk にデプロイするには

1. 作業するディレクトリに移動し、基本的な Web アプリを作成します。

   ```
   dotnet new webapp --name SimpleWebAppForBeanstalk
   ```
2. アプリケーションディレクトリに移動します。

   ```
   cd SimpleWebAppForBeanstalk
   ```
3. デプロイツールを実行します。

   ```
   dotnet aws deploy
   ```
4. Eclipse名前をAWSアプリケーションをにデプロイするにはキーを押した状態で、Enterキーを押してデフォルト名を受け入れます。既存のスタックがある場合は、を選択します。AWSアプリケーションをにデプロイするには代わりに、します。この場合、最後のオプションを選択して新規作成を押し、Enterキーを押してデフォルト名を受け入れます。
5. 次回のお問い合わせはデプロイオプションの選択で、[] オプションを選択します。AWS Elastic BeanstalkLinux 上のキーを押したまま、Enterkey。このチュートリアルでは、これがデフォルトのオプションです。

   > **Note**
   >
   > 実際のアプリケーションをデプロイするときに、ツールがプロジェクトの Dockerfile を見つけられない場合、デプロイメントツールはAWS Elastic BeanstalkLinux 上のをデフォルトのオプションとして使用します。プロジェクトに Dockerfile がある場合、ツールには別のデフォルトが表示されます。この代替シナリオの詳細については、」Amazon ECS へのウェブアプリケーションのデプロイ (p. 196)。
6. キーを押します。Enterキーをもう一度押して、アプリケーションとスタックの設定のデフォルトを受け入れ、デプロイメントを開始します。
7. デプロイプロセスが終了するのを待ちます。
8. ツールの出力の終了時には、以下の行が表示されます。「SimpleWebAppForBeanstalk.Endpointurl...")。この行には、作成された Web サイトの URL が含まれます。この URL を Web ブラウザで開くことができます。または、次に示すように、Elastic Beanstalk コンソールから作成された Web サイトを開くこともできます。
9. AWS Management Console にサインインし、Elastic Beanstalk コンソール (https://console.aws.amazon.com/elasticbeanstalk/) を開きます。

   適切なAWS必要に応じてリージョン。
10. リポジトリの []環境[] ページで、[]SimpleWebAppForBeanstalk デベロッパー環境。
11. 環境のページの上部セクションで、ヘルスステータスが であるOKをクリックし、リンクを開いて結果の Web サイトを表示します。今のところこの Web サイトを開いたままにします。

更新と再デプロイ

アプリケーションをデプロイし、結果の Web サイトを確認できたので、コンテンツにいくつかの変更を加え、アプリケーションを再デプロイします。

Web コンテンツに変更を加え、アプリケーションを再デプロイするには

1. 左Pagesサブディレクトリーを開き、Index.cshtmlテキストエディタでを使用します。
2. HTML コンテンツに変更を加え、ファイルを保存します。
3. プロジェクトのメインディレクトリで、デプロイメントツールをもう一度実行します。

```
dotnet aws deploy
```

4. Eclipseを選択します。AWSアプリケーションをにデプロイするにはで、このチュートリアルに対応するスタック名を選択し、Enterkey。このチュートリアルでは、これはです。シンプルWebAppForBeanstalkこれがデフォルトの選択肢です。
5. キーを押します。Enterキーをもう一度押して、以前と同じデフォルトを受け入れ、アプリケーションの再デプロイを待ちます。
6. 左Elastic Beanstalk コンソールなどの他の処理方法を使用したり、SimpleWebAppForBeanstalk デベロッパー環境をもう一度実行します。になっていることを確認します。ヘルスステータスが であるOKをクリックし、アプリケーションの Web サイトを更新して変更を確認します。

Cleanup

予期せぬコストを回避するため、チュートリアルの環境やアプリケーションが終了したら、必ず削除してください。

このクリーンアップは、Elastic Beanstalk コンソール (https://console.aws.amazon.com/elasticbeanstalk。

チュートリアルのアーティファクトを除去するには

1. 既存のクラウドアプリケーションのリストを取得します。

   ```
   dotnet aws list-deployments.
   ```

   リストには、このチュートリアルのデプロイメントが含まれています。シンプルWebAppForBeanstalk。
2. デプロイを削除します。

   ```
   dotnet aws delete-deployment SimpleWebAppForBeanstalk
   ```
3. 「y」と入力して削除を確認し、展開が削除されるのを待ちます。
4. 左Elastic Beanstalk コンソールなどの他の処理方法を使用したり、環境およびアプリケーションページを使用して、チュートリアルのデプロイメントが削除されたことを確認します。
5. チュートリアル中に作成した Web サイトを更新して、使用できなくなったことを確認します。

Amazon ECS へのウェブアプリケーションのデプロイ

> これはプレビューリリースの機能に関するプレリリースドキュメントです。このドキュメントは変更される可能性があります。

このチュートリアルでは、使用した Amazon ECS へのデプロイについて説明します。AWS Fargate。このチュートリアルでは、配置ツールによって提供されるデフォルトを使用します。

Prerequisites

- これで、完了です。環境のセットアップ (p. 191)およびツールのセットアップ (p. 192)。
- `-[default]`認証情報プロファイルには、必要なアクセス許可があります。
- Dockerがインストールされて実行されていますが、チュートリアルにはDockerfileは存在しません。

Deploy

アプリケーションをデプロイする準備ができたらAWS初めて、これが始める場所です。

を使用してサンプルウェブアプリケーションを Amazon ECS にデプロイするにはAWS Fargate

1. 作業するディレクトリに移動し、基本的な Web アプリを作成します。

   ```
   dotnet new webapp --name SimpleWebAppForECS
   ```
2. アプリケーションディレクトリに移動します。

   ```
   cd SimpleWebAppForECS
   ```
3. デプロイツールを実行します。

   ```
   dotnet aws deploy
   ```
4. Eclipse名前をAWSアプリケーションをにデプロイするにはキーを押した状態で、Enterキーを押してデフォルト名を受け入れます。既存のスタックがある場合は、を選択します。AWSアプリケーションをにデプロイするには代わりに、します。この場合、最後のオプションを選択して新規作成を押し、Enterキーを押してデフォルト名を受け入れます。
5. 次回のお問い合わせはデプロイオプションの選択で、[] オプションを選択します。Amazon ECS を使用する Fargate Sキーを押したまま、Enterkey。このチュートリアルでは、これは 2 番目のオプション (追加のデプロイメントオプション) であり、デフォルトではありません。

   Note

   実際のアプリケーションをデプロイするときに、デプロイメントツールがプロジェクトの Dockerfile を検出すると、Amazon ECS を使用する Fargate Sをデフォルトのオプションとして使用します。プロジェクトに Dockerfile がないが、Amazon ECS を使用する Fargate Sオプションを使用すると、Dockerfile が生成されます。
6. キーを押します。Enterキーをもう一度押して、アプリケーションとスタックの設定のデフォルトを受け入れ、デプロイメントを開始します。このチュートリアルでは、プロジェクトの Dockerfile は見つからず、Amazon ECS を使用する Fargate Sオプションが選択されている場合、ツールは Dockerfile も生成します。
7. デプロイプロセスが完了するのを待ちます。
8. ツールの出力の終了時には、以下の行が表示されます。「SimpleWebAppForecs.FargateServicesServiceUrl...」。この行には、作成された Web サイトの URL が含まれます。ウェブブラウザで URL を開き、結果の Web サイトを表示します。今のところこの Web サイトを開いたままにします。
9. ツールによって作成されたリソースを表示する場合は、Amazon ECS コンソール (https://console.aws.amazon.com/ecs/。適切なAWS必要に応じてリージョン。リポジトリの []クラスターページには、作成された新しいクラスターが表示されます。シンプルWebAppForecs。

更新と再デプロイ

アプリケーションをデプロイし、結果の Web サイトを確認できたので、コンテンツにいくつかの変更を加え、アプリケーションを再デプロイします。

Web コンテンツに変更を加え、アプリケーションを再デプロイするには

1. 左Pagesサブディレクトリーを開き、Index.cshtmlテキストエディタでを使用します。
2. HTML コンテンツに変更を加え、ファイルを保存します。
3. プロジェクトのメインディレクトリで、デプロイメントツールをもう一度実行します。

   ```
   dotnet aws deploy
   ```
4. Eclipseを選択します。AWSアプリケーションをにデプロイするにはで、このチュートリアルに対応するスタック名を選択し、Enterkey。このチュートリアルでは、これはです。シンプルWebAppForecsこれがデフォルトの選択肢です。
5. キーを押します。Enterキーをもう一度押して、以前と同じデフォルトを受け入れ、アプリケーションの再デプロイを待ちます。

6. アプリケーションの Web サイトを更新して、変更内容を確認します。

Cleanup

予期しないコストを避けるため、チュートリアルのクラスター、タスク、および ECR リポジトリの作業が終了したら、必ず削除してください。

Amazon ECS コンソール (Amazon ECS コンソール) を使用して、手動でこのクリーンアップを行うこともできます。https://console.aws.amazon.com/ecs/。

チュートリアルのアーティファクトを除去するには

1. 既存のクラウドアプリケーションのリストを取得します。

   ```
   dotnet aws list-deployments
   ```

   リストには、このチュートリアルのデプロイメントが含まれています。シンプルWebAppForecs。
2. デプロイを削除します。

   ```
   dotnet aws delete-deployment SimpleWebAppForECS
   ```
3. 「y」と入力して削除を確認し、展開が削除されるのを待ちます。
4. 左Amazon ECS コンソールなどの他の処理方法を使用したり、クラスターおよびタスク定義ページを使用して、チュートリアルのデプロイメントが削除されたことを確認します。
5. （オプション）Amazon ECR,リポジトリページで、チュートリアル用に作成されたリポジトリーを削除できます。`simplewebappforecs`
6. チュートリアル中に作成した Web サイトを更新して、使用できなくなったことを確認します。

変更後のアプリケーションの再デプロイ

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。

アプリケーションを変更した場合は、デプロイツールを使用して再デプロイできます。これを行うには、アプリケーションのプロジェクトディレクトリに移動し、`dotnet aws deploy`をクリックします。

再配置の特定の時点で、ツールによってを選択します。AWSアプリケーションをにデプロイするには。デプロイスタックのリストをスキャンして、次のいずれかを実行します。

- .NET アプリケーションに対応するスタック名を選択します。

  これを行うと、アプリケーションの最後のデプロイメントの設定が表示されます。一部の詳細設定のみを変更できます。残りは読み取り専用です。
- 最後のオプション新規作成[] をクリックし、新しい名前を入力します。

  これを行うと、ツールは新しいデプロイスタックを作成し、設定を使用して何をしたいのかを決定できます (デフォルトを受け入れるか、変更を加えます)。

配置ツールで使用できる配置設定

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。

デプロイまたは再デプロイのステップの 1 つは、使用するコンピューティングサービスを選択することです。これは、多くの場合、デプロイメントツールのデフォルトの提案です。目的の展開オプションを選択すると、そのオプションで使用できる共通設定の一覧がツールに表示されます。詳細設定も利用できます。詳細設定には、」more「コマンドプロンプトで。

多くの設定は自明です。以下の設定はあまり分かりません。

へのデプロイ AWS Elastic Beanstalk

- アプリケーション IAM ロール

  -アプリケーション IAM ロール提供AWSにデプロイされたアプリケーションの認証情報AWSはユーザーに代わって処理します。IAM ロールを使用すると、アプリケーションはAWSAmazon S3 バケットのようなリソース。

  この設定を使用して、既存の IAM ロールから選択するか、ツールがアプリケーション専用のロールを作成できるようにすることができます。これはデフォルトの動作です。既存の IAM ロールの定義を確認するには、次の IAM コンソールを開きます。https://console.aws.amazon.com/iam/を選択し、ロールページで.

- EC2 インスタンスタイプ

  この設定を使用すると、ツールが示唆するインスタンスタイプ以外の Amazon EC2 インスタンスタイプを指定できます。Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/を選択し、インスタンスタイプページの下のインスタンス。EC2 インスタンスタイプの詳細については、「」を参照してください。https://aws.amazon.com/ec2/instance-types/。

- キーペア

  この設定のデフォルトは「EMPTY」(key pair なし) です。これは、EC2 インスタンスに SSH 接続できないことを意味します。EC2 インスタンスに SSH 接続する場合は、リストからkey pair を選択できます。リスト内のキーペアを確認するには、EC2 コンソールを選択し、キーペアページの下のネットワークとセキュリティ。このツールで新しいkey pair を作成することもできます。このオプションを選択する場合は、key pair 名前と、秘密キーが格納されるディレクトリを入力します。

  > Warning
  >
  > 新しいkey pair を作成してコンピュータに保存する場合は、適切な予防措置を講じてキーペアを保護してください。

を使用した Amazon ECS へのデプロイ AWS Fargate

- アプリケーション IAM ロール

  -アプリケーション IAM ロール提供AWSにデプロイされたアプリケーションの認証情報AWSはユーザーに代わって処理します。IAM ロールを使用すると、アプリケーションはAWSAmazon S3 バケットのようなリソース。

  この設定を使用して、既存の IAM ロールから選択するか、ツールがアプリケーション専用のロールを作成できるようにすることができます。これはデフォルトの動作です。既存の IAM ロールの定義を確認するには、次の IAM コンソールを開きます。https://console.aws.amazon.com/iam/を選択し、ロールページで. IAM ロールがリストされていない場合、適切な IAM ロールが存在しないことを意味します。したがって、ツールがロールを作成する唯一の選択肢です。

- Virtual Private Cloud

  この設定のデフォルト値はアカウントの Default VPC です。このアカウントは、https://console.aws.amazon.com/vpc/を選択し、VPC。アカウントのデフォルト VPC にははいのデフォルトVPC列VPC。その列を表示するには、水平にスクロールする必要があります。別の VPC を選択するか、ツールによってアプリケーションの新しい VPC を作成することができます。

設定ファイルの操作

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。

実行すると、`deploy`コマンドのAWS.NET 配置ツールでは、ツールからのプロンプトに応答して配置オプションを選択できます。

また、JSON 設定ファイルを指定することもできます。`--apply`コマンドの [] オプション。ツールは、ファイルで指定されている JSON パラメータからデプロイメントオプションを読み取り、プロンプトでこれらのオプションを使用します。指定されていないパラメータにはデフォルト値を使用します。

JSON 設定ファイルを`-s`(`--silent`) オプションを使用すると、デプロイメントツールにまったくプロンプトが表示されることなく、デプロイメントオプションを指定できます。

このセクションでは、設定ファイルの作成に使用する JSON 定義と構文を定義します。

> Note
>
> JSON 設定ファイルのSupport は、デプロイメントツールのバージョン 0.11.16 で追加されました。

共通 JSON パラメータ

JSON 設定ファイルに含めることができるパラメーターは、このトピックで後述するように、実行するデプロイの種類によって異なります。次のパラメータは、すべてのデプロイの種類に共通です。

AWSプロファイル

: の名前。AWS使用しない場合に使用するプロファイルを`[default]`プロファイル。

AWSRegion

: の名前。AWSからリージョンを使用していない場合に使用するリージョン`[default]`プロファイル。

StackName

: の名前。AWS CloudFormationスタックを使用して、アプリケーションに使用することができます。これは、既存のスタックの名前、または作成する新しいスタックの名前にすることができます。

Amazon ECS デプロイ用の JSON 構文

```
{
    "AWSProfile": "string",
    "AWSRegion": "string",
    "StackName": "string",
    "RecipeId": "AspNetAppEcsFargate" | "ConsoleAppEcsFargateService" |
 "ConsoleAppEcsFargateScheduleTask",
    "OptionSettingsConfig":
    {
        "ECSCluster":
        {
            "CreateNew": boolean,
            "NewClusterName": "string" | "ClusterArn": "string"
        },
        "ECSServiceName": "string",
        "DesiredCount": integer,
```

```
        "ApplicationIAMRole":
        {
            "CreateNew": boolean,
            "RoleArn": "string"
        },
        "Schedule": "string",
        "Vpc":
        {
            "IsDefault": boolean,
            "CreateNew": boolean | "VpcId": "string"
        },
        "AdditionalECSServiceSecurityGroups": "string",
        "ECSServiceSecurityGroups": "string",
        "TaskCpu": integer,
        "TaskMemory": integer,
        "DockerExecutionDirectory": "string"
    }
}
```

以下のパラメータ定義は、Amazon ECS デプロイの JSON 構文に固有です。また、「the section called "共通 JSON パラメータ" (p. 200)」も参照してください。

レシペID

実行する Amazon ECS デプロイのタイプを識別する値。ASP.NET Core ウェブアプリケーション、長時間実行されるサービスアプリケーション、またはスケジュールされたタスク。

オプション設定設定

Amazon ECS デプロイを設定するには、以下のオプションを使用できます。

ECSCluster

展開に使用する ECS クラスター。これは、新しいクラスタ (デフォルト) または既存のクラスタ (デフォルト) です。このパラメータが存在しない場合は、プロジェクトと同じ名前の新しいクラスターが作成されます。新しいクラスターに名前を付けたい場合は、`NewClusterName`。既存のクラスターを使用している場合は、`CreateNew` ~ `false`に追加し、クラスターの ARN を`ClusterArn`。

ECSサービス名

このパラメータは、に対してのみ有効です。`AspNetAppEcsFargate`および`ConsoleAppEcsFargateService`レシピからアクセス可能になっています。

クラスタで実行されている ECS サービスの名前。このパラメータが存在しない場合、サービスの名前は「<YourProjectName>-service」になります。

DesiredCount

このパラメータは、に対してのみ有効です。`AspNetAppEcsFargate`および`ConsoleAppEcsFargateService`レシピからアクセス可能になっています。

サービスに対して実行する ECS タスクの数。指定した場合、値は 1 以上、5000 以上でなければなりません。デフォルトは 3 です。`AspNetAppEcsFargate`レシピと 1 の`ConsoleAppEcsFargateService`レシピ。

アプリケーションIAmRole

提供する IAM ロールAWS認証情報をアプリケーションに追加してアクセスするAWSのサービス。新しいロール (デフォルト) を作成することも、既存のロールを使用することもできます。既存のロールを使用するには、`CreateNew` ~ `false`ロールの ARN をに組み込みます。`RoleArn`。

スケジュール

このパラメータは、に対してのみ有効です。`ConsoleAppEcsFargateScheduleTask`レシピ。

Amazon CloudWatch Events タスクを実行するタイミングを決めるスケジュールまたは割合 (頻度)。この値の形式の詳細については、ルールのスケジュール式のAmazon CloudWatch Events ユーザーガイド。

Vpc

アプリケーションを起動する Amazon Virtual Private Cloud (VPC)。デフォルト VPC (デフォルトの動作) 、新しい VPC、または既存の VPC を指定できます。新しい VPC を作成するには、`IsDefault~false`および`CreateNew~true`。既存の VPC を使用するには、`IsDefault~false`をクリックし、VPC ID を`VpcId`。

追加サービスサービスサービスサービスグループ

このパラメータは、に対してのみ有効です。`AspNetAppEcsFargate`レシピ。

ECS サービスに割り当てる EC2 セキュリティグループのコンマ区切りリスト。

ECSサービスセキュリティグループ

このパラメータは、に対してのみ有効です。`ConsoleAppEcsFargateService`レシピ。

ECS サービスに割り当てる EC2 セキュリティグループのコンマ区切りリスト。

タスクCPU

タスクで使用されている CPU ユニットの数。有効な値は「256」(デフォルト)、「512」、「1024」、「2048」、および「4096」です。詳細については、「」を参照してください。Amazon ECSAWS FargateのAmazon Elastic Container Service, 具体的には,タスク CPU とメモリ。

タスクメモリ

タスクが使用するメモリの量 (MB 単位)。有効な値は「512」(デフォルト)、「1024」、「2048」、「3072」、「4096」、「5120」、「6144」、「7168」、「8192」、「9216」、「10240」、「11264」、「12288」、「13312」、「14336」、「15360」、「1638です4"、" 17408"、"18432"、" 19456"、"20480"、" 21504"、"22528"、" 23552"、"24576"、" 25600"、"26624"、" 27648"、"28672"、" 29696"、"30720"。詳細については、「」を参照してください。Amazon ECSAWS FargateのAmazon Elastic Container Service, 具体的には,タスク CPU とメモリ。

ドックエクセクションディレクトリ

Dockerを使用している場合は、オペレーティングシステム用に適切にエスケープされた文字列としてフォーマットされたDocker実行環境へのパス (たとえば、Windows : 「C : \\ codebase") 。

JSON 構文AWS Elastic Beanstalkデプロイ

```
{
    "AWSProfile": "string",
    "AWSRegion": "string",
    "StackName": "string",
    "RecipeId": "AspNetAppElasticBeanstalkLinux",
    "OptionSettingsConfig":
    {
        "BeanstalkApplication":
        {
            "CreateNew": boolean,
            "ApplicationName": "string"
        },
        "EnvironmentName": "string",
        "InstanceType": "string",
        "EnvironmentType": "SingleInstance" | "LoadBalanced",
        "LoadBalancerType": "application" | "classic" | "network",
        "ApplicationIAMRole":
```

```
        {
            "CreateNew": boolean,
            "RoleArn": "string"
        },
        "EC2KeyPair": "string",
        "ElasticBeanstalkPlatformArn": "string",
        "ElasticBeanstalkManagedPlatformUpdates":
        {
            "ManagedActionsEnabled": boolean,
            "PreferredStartTime": "Mon:12:00",
            "UpdateLevel": "minor"
        }
    }
}
```

次のパラメータ定義は、Elastic Beanstalk デプロイメントの JSON 構文に固有です。また、「the section called “共通 JSON パラメータ” (p. 200)」も参照してください。

レシペID

タイプを識別する値AWS Elastic Beanstalk配置を使用します。この場合は、ASP.NET Core Web アプリです。

オプション設定設定

Elastic Beanstalk デプロイメントを設定するには、次のオプションを使用できます。

BeanStalkアプリケーション

Elastic Beanstalk アプリケーションの名前。これは、新しいアプリケーション (デフォルト) または既存のアプリケーション。名前をから指定しない場合`ApplicationName`場合、アプリケーションはプロジェクトと同じ名前になります。

EnvironmentName

の名前。Elastic Beanstalk 環境アプリケーションを実行するには。このパラメータが存在しない場合、環境の名前は「<YourProjectName>-dev」になります。

InstanceType

-Amazon EC2 インスタンスタイプ (例 : 「t2.micro」) 。このパラメータが含まれていない場合、プロジェクトの要件に基づいてインスタンスタイプが選択されます。

環境タイプ

-Elastic Beanstalk 環境のタイプを作成します。開発作業のための単一のインスタンス (デフォルト) または実稼働のための負荷分散です。

ロード・バランサー・タイプ

-ロードバランサーのタイプクラシック、アプリケーション (デフォルト) 、またはネットワークなど、環境に適したものを使用します。

このパラメータは、値が`EnvironmentType`は「LoadBalanced」です。

アプリケーションIAmRole

提供する IAM ロールAWS認証情報をアプリケーションに追加してアクセスするAWSのサービス。新しいロール (デフォルト) を作成することも、既存のロールを使用することもできます。既存のロールを使用するには、`CreateNew`~`false`ロールの ARN をに組み込みます。`RoleArn`。

EC2KeyPair

Elastic Beanstalk 環境で EC2 インスタンスに SSH 接続するために使用できる EC2 key pair。このパラメータを指定せず、デプロイツールの実行中にkey pair をインタラクティブに選択しないと、EC2 インスタンスに SSH 接続できなくなります。

ElasticbeanstalkPlatformarn

の ARNAWS Elastic Beanstalkプラットフォームを環境で使用します。このパラメータが存在しない場合は、最新の Elastic Beanstalk プラットフォームの ARN が使用されます。

この ARN を作成する方法については、「」を参照してください。ARN 形式のAWS Elastic Beanstalkデベロッパーガイド。

ElasticBeanstalkManagedプラットフォームの更新

このパラメータを使用して、Elastic Beanstalk プラットフォームの自動更新をマネージドプラットフォーム更新で説明されている。AWS Elastic Beanstalkデベロッパーガイド。もしManagedActionsEnabledは、 に設定されます。true (デフォルト) では、毎週のメンテナンス期間をPreferredStartTimeに設定されています。デフォルトは「Sun: 00:00」です。さらに、UpdateLevelをクリックして、適用するパッチレベル (「マイナー」(デフォルト) または「パッチ」) を指定します。これらのオプションについては、マネージドアクションオプションの名前空間『Elastic Beanstalk 開発者ガイド』を参照してください。

更新プロセス中も、アプリケーションは引き続き利用できます。

Blazor WebAssembly アプリデプロイメント用の JSON 構文

```
{
    "AWSProfile": "string",
    "AWSRegion": "string",
    "StackName": "string",
    "RecipeId": "BlazorWasm",
    "OptionSettingsConfig":
    {
        "IndexDocument": "string",
        "ErrorDocument": "string",
        "Redirect404ToRoot": boolean
    }
}
```

Note

このデプロイタスクは、Blazor WebAssembly アプリケーションを Amazon S3 バケットにデプロイします。デプロイ中に作成されたバケットはウェブホスティング用に設定され、その内容は読み取りアクセスで一般に公開されます。

以下のパラメーター定義は、Blazor WebAssembly デプロイメントの JSON 構文に固有のものです。また、「the section called “共通 JSON パラメータ” (p. 200)」も参照してください。

レシペID

実行するデプロイメントの種類を識別する値。この場合は Blazor WebAssembly アプリケーション。

オプション設定設定

Blazor WebAssembly デプロイを設定するには、以下のオプションを使用できます。

IndexDocument

WebAssembly アプリのエンドポイントにリソースパスなしでアクセスされるときに使用する Web ページの名前。デフォルトのページ名はindex.html。

ErrorDocument

リソースパスへのアクセス中にエラーが発生したときに使用する Web ページの名前です。デフォルト値は空の文字列です。

リダイレクト404toRoot

このパラメータの設定`true`(デフォルト) の場合、404 となるリクエストは Web アプリケーションのインデックスドキュメントにリダイレクトされます。これは`IndexDocument`。

のセキュリティAWS製品またはサービス

クラウドセキュリティはアマゾン ウェブ サービス (AWS) の最優先事項です。AWS のお客様は、セキュリティを最も重視する組織の要件を満たす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られます。セキュリティは、AWS とお客様の間の共有責任です。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 は、AWS クラウド内でサービスを実行するインフラストラクチャを保護する責任を担い、安全に使用できるサービスを提供します。AWS ではセキュリティに対する責任が最優先事項であり、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人によって定期的にテストおよび検証されています。

クラウド内のセキュリティ – お客様の責任は、使用している AWS のサービスや、データの機密性、組織の要件、適用される法律や規制などのその他の要因によって決まります。

このAWS製品またはサービスは、責任共有モデル特定のAmazon Web Services 通じて (AWS) サポートしているサービス。AWS サービスのセキュリティ情報については、AWS のサービスのセキュリティに関するドキュメントページと、AWS コンプライアンスプログラムによる AWS 対象範囲内のサービスを参照してください。

トピック

この中のデータ保護AWS製品またはサービス

-AWS 責任共有モデルこれは、データ保護に適用されます。AWS製品またはサービス。このモデルで説明されているように、AWS は、 AWS クラウド のすべて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を担います。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。このコンテンツには、使用する AWS のサービスのセキュリティ設定および管理タスクが含まれます。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の「The AWS Shared Responsibility Model and GDPR」を参照してください。

データを保護するため、 AWS アカウント の認証情報を保護し、AWS Identity and Access Management (IAM) を使用して個々のユーザーアカウントをセットアップすることをお勧めします。この方法により、それぞれの職務を遂行するために必要なアクセス許可のみを各ユーザーに付与できます。また、以下の方法でデータを保護することをお勧めします。

- 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 以降が推奨されています。
- AWS CloudTrail で API とユーザーアクティビティログをセットアップします。

- AWS 暗号化ソリューションを、AWS のサービス内のすべてのデフォルトのセキュリティ管理と一緒に使用します。
- 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。これにより、Amazon S3 に保存される個人データの検出と保護が支援されます。
- コマンドラインインターフェイスまたは API を使用して AWS 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。使用可能な 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。

顧客の E メールアドレスなどの機密情報やセンシティブ情報は、タグや [名前] フィールドなどの自由形式の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、この作業を行う場合も同様です。AWS製品、サービス、またはその他AWSコンソール、API、を使用したサービスAWS CLI, またはAWSSDK。タグまたは名前に使用する自由形式のフィールドに入力したデータは、請求ログまたは診断ログに使用できます。外部サーバーへの URL を指定する場合は、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 に含めないことを強くお勧めします。

この Identity and Access ManagementAWS製品またはサービス

AWS Identity and Access Management(IAM) はAmazon Web Services (AWS) サービスで、管理者がへのアクセスを安全にコントロールするのに役立ちます。AWSリソースの使用料金を見積もることができます。IAM 管理者は、誰にできるかを制御する認証済み(サインイン) と承認済みリソースを使用するには (権限を持つ)AWSのサービス。IAM は、AWS のサービスで追加料金は発生しません。

これを使うにはAWSアクセスする製品またはサービスAWSの場合は、AWSアカウントとAWS認証情報。あなたのセキュリティを高めるためにAWSアカウントの場合は、IAM ユーザーを使用する代わりにアクセス認証情報を提供するにはAWSアカウント認証情報。

IAM の使用の詳細については、「」AWS Identity and Access Management。

IAM ユーザーの概要と、IAM ユーザーがアカウントのセキュリティにとって重要である理由については、「」AWSセキュリティの認証情報のAmazon Web Services 全般リファレンス。

このAWS製品またはサービスは、責任共有モデル特定のAmazon Web Services 通じて (AWS) サポートしているサービス。AWS サービスのセキュリティ情報については、AWS のサービスのセキュリティに関するドキュメントページと、AWS コンプライアンスプログラムによる AWS 対象範囲内のサービスを参照してください。

このコンプライアンス検証AWS製品またはサービス

このAWS製品またはサービスは、責任共有モデル特定のAmazon Web Services 通じて (AWS) サポートしているサービス。AWS サービスのセキュリティ情報については、AWS のサービスのセキュリティに関するドキュメントページと、AWS コンプライアンスプログラムによる AWS 対象範囲内のサービスを参照してください。

AWS サービスのセキュリティおよびコンプライアンスは、複数の 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り評価されます。これには、SOC、PCI、FedRAMP、HIPAA、その他が含まれます。AWS は特定のコンプライアンスプログラムの対象となる AWS のサービスのリストを「コンプライアンスプログラムによる AWS 対象範囲内のサービス」で提供し、頻繁に更新しています。

サードパーティーの監査レポートは、AWS Artifact を使用してダウンロードできます。詳細については、「AWS Artifact にレポートをダウンロードする」を参照してください。

AWS コンプライアンスプログラムの詳細については、「AWS コンプライアンスプログラム」を参照してください。

これを使用する際のコンプライアンスの責任AWSにアクセスするための製品またはサービスAWSサービスは、組織のデータの機密性や組織のコンプライアンス目的、適用可能な法律、規制によって決定されます。AWS のサービスの使用が HIPAA、PCI、または FedRAMP などの規格に準拠していることを前提としている場合、AWS は以下を支援するリソースを提供します。

- セキュリティおよびコンプライアンスのクイックスタートガイド – 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を AWS にデプロイするための手順を説明するデプロイメントガイド。
- HIPAA Security and Compliance for Architecting for HIPA— 企業の使用方法を説明したホワイトペーパーAWSをクリックして HIPAA に準拠するアプリケーションを作成します。
- AWS コンプライアンスのリソース – お客様の業界や場所に適用される可能性があるワークブックとガイドのコレクション。
- AWS Config – 自社プラクティス、業界ガイドライン、および規制に対するリソースの設定の準拠状態を評価するサービス。
- AWS Security Hub – セキュリティに関する業界標準およびベストプラクティスへの準拠を確認するのに役立つ、AWS 内でのセキュリティ状態を包括的に表示したもの。

の耐障害性AWS製品またはサービス

アマゾン ウェブ サービス (AWS) のグローバルインフラストラクチャは AWS リージョンとアベイラビリティーゾーンを中心として構築されます。

AWS リージョンには、低レイテンシー、高いスループット、そして高度の冗長ネットワークで接続されている複数の物理的に独立· 隔離されたアベイラビリティーゾーンがあります。

アベイラビリティーゾーンでは、アベイラビリティー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、耐障害性、および拡張性が優れています。

AWS リージョンとアベイラビリティーゾーンの詳細については、「AWS グローバルインフラストラクチャ」を参照してください。

このAWS製品またはサービスは、責任共有モデル特定のAmazon Web Services 通じて (AWS) サポートしているサービス。AWS サービスのセキュリティ情報については、AWS のサービスのセキュリティに関するドキュメントページと、AWS コンプライアンスプログラムによる AWS 対象範囲内のサービスを参照してください。

このためのインフラストラクチャセキュリティ AWS製品またはサービス

このAWS製品またはサービスは、責任共有モデル特定のAmazon Web Services 通じて (AWS) サポートしているサービス。AWS サービスのセキュリティ情報については、AWS のサービスのセキュリティに関するドキュメントページと、AWS コンプライアンスプログラムによる AWS 対象範囲内のサービスを参照してください。

これでTLS 1.2 の適用AWS製品またはサービス

との通信時にセキュリティを強化するにはAWSサービスでは、これを設定する必要がありますAWSTLS 1.2 以降を使用する製品またはサービス。

AWS SDK for .NET は、基盤となる .NET ランタイムを使用して、適用するセキュリティプロトコルを決定します。現行バージョンの .NET は、デフォルトでオペレーティングシステムがサポートする最新の設定済みプロトコルを使用します。この SDK の動作はアプリケーションで上書きできますが、上書きは推奨されません。

.NET Core

.NET Core は、デフォルトでオペレーティングシステムがサポートする最新の設定済みプロトコルを使用します。AWS SDK for .NET は、この動作を上書きする機構を提供していません。

バージョン 2.1 より前の .NET Core を使用している場合は、.NET Core バージョンをアップグレードすることを強くお勧めします。

オペレーティングシステムごとの固有の情報については、以下を参照してください。

Windows

Windows の最新のディストリビューションでは、TLS 1.2 のサポートが デフォルトで有効になっています。Windows 7 SP1 または Windows Server 2008 R2 SP1 上で実行している場合は、レジストリで TLS 1.2 のサポートが有効になっていることを確認する必要があります。詳細については、https://docs.microsoft.com/ja-jp/windows-server/security/tls/tls-registry-settings#tls-12 を参照してください。以前のディストリビューションを実行している場合は、オペレーティングシステムをアップグレードする必要があります。

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 は以下のようになります。OS X Mavericks v10.9以降。.NET Core バージョン 2.1 以降では、で説明しているように、より新しいバージョンの macOS が必要です。https://docs.microsoft.com/en-us/dotnet/core/install/dependencies?tabs=netcore21&pivots=os-macos。

.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 の バージョン 1.0.1 (2012 年 3 月 14 日) には、TLS 1.2 のサポートが追加されました。

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 の バージョン 1.0.1 (2012 年 3 月 14 日) には、TLS 1.2 のサポートが追加されました。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。

.NET Framework

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の設定を使用しない .NET Framework ランタイム (.NET Framework 3.5 ~ 4.5.2) 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対して 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

アプリケーションを Windows 7 SP1 または Windows Server 2008 R2 SP1 上の新しい .NET Framework で実行している場合は、レジストリで TLS 1.2 のサポートが有効になっていることを確認する必要があります。詳細については、https://docs.microsoft.com/ja-jp/windows-server/security/tls/tls-registry-settings#tls-12 を参照してください。新しいバージョンの Windows の場合、これはデフォルトで有効になっています。

.NET Framework で TLS を使用するための詳細なベストプラクティスについては、Microsoft の記事 (https://docs.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 のサービスに対するすべての呼び出しに AWS SDK for .NET を使用します。環境の動作は、次のように、実行している Windows PowerShell のバージョンによって異なります。

Windows PowerShell 2.0 ~ 5.x

Windows PowerShell 2.0 ~ 5.x は .NET Framework で実行します。どの .NET ランタイム (2.0 または 4.0) が PowerShell で使用されているかは、次のコマンドで確認できます。

```
$PSVersionTable.CLRVersion
```

- .NET ランタイム 2.0 を使用している場合は、AWS SDK for .NET および .NET Framework 3.5 に関する前述の手順に従います。
- .NET ランタイム 4.0 を使用している場合は、AWS SDK for .NET および .NET Framework 4+ に関する前述の手順に従います。

Windows PowerShell 6.0

Windows PowerShell 6.0 以降は、.NET Core で実行します。どのバージョンの .NET Core が使用されているかは、次のコマンドで確認できます。

```
[System.Reflection.Assembly]::GetEntryAssembly().GetCustomAttributes([System.Runtime.Versioning.TargetF
 $true).FrameworkName
```

AWS SDK for .NET および該当するバージョンの .NET Core に関する前述の手順に従います。

Xamarin

Xamarin については、https://docs.microsoft.com/ja-jp/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 のオプション: HttpClient の実装をに設定する必要があります。AndroidSSL/TLSの実装を次のように設定します。Native TLS 1.2。

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: HttpClient の実装をに設定する必要があります。nsurlSession。

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,をビルドする,Mac ビルド: HttpClient の実装をに設定する必要があります。nsurlSession。

Unity

Unity 2018.2 以降を使用し、.NET 4.x Equivalent スクリプティングランタイムを使用する必要があります。これを設定するには、[Project Settings]、[Configuration]、[Player] の順に選択します。詳細については、https://docs.unity3d.com/2019.1/Documentation/Manual/ScriptingRuntimeUpgrade.html を参照してください。.NET 4.x Equivalent スクリプティングランタイムは、Mono または IL2CPP を実行するすべての Unity プラットフォームで TLS 1.2 のサポートを有効にします。詳細については、https://blogs.unity3d.com/2018/07/11/scripting-runtime-improvements-in-unity-2018-2/ を参照してください。

ブラウザ (Blazor WebAssembly 用)

WebAssembly は、サーバー上ではなくブラウザで実行され、ブラウザを使用して HTTP トラフィックを処理します。したがって、TLS のサポートはブラウザのサポートによって決まります。

Blazor WebAssembly は (ASP.NET Core 3.1 のプレビューで) 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。詳細については、https://docs.microsoft.com/en-us/aspnet/core/blazor/supported-platforms。すべてのメインストリームブラウザは、WebAssembly をサポートする前に TLS 1.2 をサポートしていました。これに該当するブラウザの場合、アプリを実行すると、アプリは TLS 1.2 経由で通信できます。

詳細と検証については、ブラウザのドキュメントを参照してください。

Amazon S3 暗号化クライアントの移行

このトピックでは、Amazon シンプルストレージサービス (Amazon S3) 暗号化クライアントのバージョン 1 (V1) からバージョン 2 (V2) にアプリケーションを移行し、移行プロセス全体でアプリケーションの可用性を確保する方法について説明します。

V2 クライアントで暗号化されたオブジェクトは、V1 クライアントで復号化できません。すべてのオブジェクトを一度に再暗号化することなく、新しいクライアントへの移行を容易にするために、「V1-Transitional」クライアントが提供されています。このクライアントは次のことができます復号V1 および V2 で暗号化されたオブジェクトの両方ですが、暗号化するV1 互換形式のオブジェクトのみです。V2 クライアントは次の操作を実行できます。復号V1 および V2 で暗号化されたオブジェクトの両方 (V1 オブジェクトに対して有効になっている場合) が、暗号化するV2 互換形式のオブジェクトのみです。

移行の概要

この移行は 3 つのフェーズで行われます。これらのフェーズについては、ここで紹介し、後で詳しく説明します。各フェーズを完了する必要がありますすべて次のフェーズが開始される前に共有オブジェクトを使用するクライアント。

1. 既存のクライアントを V1 移行クライアントに更新して、新しいフォーマットを読み込みます。まず、V1 クライアントではなく V1 移行クライアントに依存するようにアプリケーションを更新しま

す。V1-Transitional クライアントを使用すると、既存のコードで、新しい V2 クライアントによって書き込ま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復号化できます。

Note

V1 移行クライアントは、移行目的でのみ提供されます。V1 移行クライアントに移行した後、V2 クライアントへのアップグレードに進みます。

2. V1 移行クライアントを V2 クライアントに移行して、新しいフォーマットを作成します。次に、アプリケーション内のすべての V1 移行クライアントを 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`。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、それらのクライアントは V1 互換形式で暗号化されたオブジェクトを復号化できます。
3. V2 クライアントを更新して V1 形式を読み込まないようにします。最後に、すべてのクライアントが V2 に移行され、すべてのオブジェクトが V2 互換形式で暗号化または再暗号化された後、V2 セキュリティプロファイルを`V2`の代わりに`V2AndLegacy`。これにより、V1 互換形式のオブジェクトの復号化を防ぐことができます。

既存のクライアントを V1 移行クライアントに更新して新しいフォーマットを読み込む

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 移行クライアントを使用すると、アプリケーションは V1 および V2 で暗号化されたオブジェクトの両方を復号化できます。このクライアントはAmazon.Extensions.S3.EncryptionNuGet パッケージです。V1-Transitional 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. 新しい依存関係を築くAmazon.Extensions.S3.Encryptionパッケージ。プロジェクトが直接依存している場合AWSSDK.S3またはawssdk.keyManagementサービスパッケージの場合は、それらの依存関係を更新するか、更新され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、それらの依存関係を削除する必要があります。
2. 適切なものを変更する`using`からの声明`Amazon.S3.Encryption`~`Amazon.Extensions.S3.Encryption`のように、次のようになります。

```
// 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 移行クライアントは V1 クライアントと完全に API 互換性があるため、他のコードの変更は必要ありません。

V1 移行クライアントを V2 クライアントに移行して新しいフォーマットを書き込む

V2 クライアントはAmazon.Extensions.S3.EncryptionNuGet パッケージです。これにより、アプリケーションは V1 および V2 で暗号化されたオブジェクトの両方を復号化できますが (そのように構成されている場合) 、オブジェクトは V2 互換形式でのみ暗号化されます。

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込むと、V2 暗号化および復号化クライアントにアプリケーションを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を宣言します。つまり、`KmsType.KmsContext`。
      iii. KMS にこのデータキーに関連付ける暗号化コンテキストを指定します。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)。ただし、追加のコンテキストを提供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 (対称暗号化または非対称暗号化) を使用する場合:
      i. 以下の内容を指定します。`AES`または`RSA`暗号化マテリアルを含むインスタンス。
      ii. 使用する暗号化アルゴリズムを宣言する。つまり、`SymmetricAlgorithmType.AesGcm`または`AsymmetricAlgorithmType.RsaOaepSha1`。
2. 変更`AmazonS3CryptoConfiguration`～`AmazonS3CryptoConfigurationV2`と`SecurityProfile`プロパティの設定は次のとおりです。`SecurityProfile.V2AndLegacy`。
3. `AmazonS3EncryptionClient` を `AmazonS3EncryptionClientV2` に変更します。このクライアントは、新しく変換された`AmazonS3CryptoConfigurationV2`および`EncryptionMaterialsV2`前のステップからのオブジェクト。

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`のいずれかです`ObjectMetadata`または`InstructionFile`。

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から変更されていないことを確認するためです。

例: 非対称アルゴリズム (RSA から RSA-OAEP-SHA1 キーラップ)

`StorageMode`のいずれかです`ObjectMetadata`または`InstructionFile`。

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 V1 形式を読み込まないように更新する

最終的に、すべてのオブジェクトは 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では、V2 クライアントの V1 互換性を無効にするには、`SecurityProfile`プロパティに`SecurityProfile.V2`をに設定しています。

```
//var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy);
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2);
```

に関する特別な考慮事項AWS SDK for .NET

このセクションでは、通常の構成または手順が適切でない、または十分ではない特殊なケースに関する考慮事項について説明します。

トピック

のアセンブリの取得AWS SDK for .NET

このトピックでは、AWSSDK アセンブリを取得して、プロジェクトで使用するためにローカル (またはオンプレミス) に格納する方法について説明します。これはないSDK参照を処理するための推奨される方法ですが、一部の環境では必要です。

> Note
>
> SDK リファレンスを処理するための推奨される方法は、各プロジェクトに必要な NuGet パッケージだけをダウンロードしてインストールすることです。この方法については、NuGet を使用した AWSSDK パッケージのインストール (p. 29)。

プロジェクト単位で 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、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します。

(これは推奨される方法 (p. 29)への参照を処理するためのAWS SDK for .NET。)

1. 次の ZIP ファイルのいずれかをダウンロードします。
   - aws-sdk-ネット標準2.0.zip
   - aws-sdk-net45.zip
   - aws-sdk-net35.zip
2. アセンブリをファイルシステム上のいくつかの「ダウンロード」フォルダに抽出します。どこかは関係ありません。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するときに、このフォルダから必要なアセンブリを取得します。NuGet を使わずに AWSSDK アセンブリをインストールする (p. 30)。

MSIをWindowsにインストールする

(これは推奨される方法 (p. 29)への参照を処理するためのAWS SDK for .NET。)

NuGet を使用する代わりに MSI をインストールする必要がある場合は、legacyMSI の MSIhttps://sdk-for-net.amazonwebservices.com/latest/AWSToolsAndSDKForNet.msi。

デフォルトでは、AWS SDK for .NETがインストールされているProgram Files管理者権限が必要なフォルダです。管理者以外として SDK をインストールするには、別のフォルダを選択します。

アプリケーション内の認証情報およびプロファイルへのアクセス

認証情報を使用するための推奨の方法は、AWS SDK for .NETを使用してそれらを見つけて取得します。詳細については、」認証情報とプロファイルの解決 (p. 25)。

ただし、プロファイルと認証情報をアクティブに取得して、それら認証情報を明示的に使用するように、アプリケーションを設定することもできます。AWSサービスクライアント。

プロファイルと資格情報をアクティブに取得するには、Amazon.cruntime.CredentialManagement名前空間。

- ファイルを使用して、プロファイルを見つけるにはAWS認証情報ファイル形式 (共有AWS認証情報ファイルをデフォルトの場所に保存する (p. 21)またはカスタム認証情報ファイル) を使用する場合は、SharedCredentialsFileクラス。この形式のファイルは、単に認証情報ファイル簡潔にするために、このテキストで。
- SDK ストアでプロファイルを検索するには、NetSDKCredentialsFileクラス。
- クラスプロパティの設定に応じて、認証情報ファイルと SDK ストアの両方を検索するには、資格情報プロファイルチェーンクラス。

  このクラスを使用して、プロファイルを検索できます。このクラスを使用して、AWS認証情報を直接使用するのではなく、AWSCredentialsFactoryクラス (次に説明します) 。
- プロファイルからさまざまな種類の認証情報を取得または作成するには、Awscreentialsファクトリークラス。

以下のセクションでは、これらのクラスの例を示します。

クレデンシャルプロファイルチェーンクラスの例

認証情報またはプロファイルは、資格情報プロファイルチェーンクラスを使用して、TryGetAWSCredentialsまたはTryGetProfileメソッド。-ProfilesLocationプロパティにより、以下のように、メソッドの動作が決まります。

- もしProfilesLocationが null または空の場合は、プラットフォームがサポートしている場合に SDK ストアを検索し、共有AWS認証情報ファイルをデフォルトの場所に保存します。
- そのファイルに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 ストアまたは共有のAWS認証情報ファイル

この例では、使用して、認証情報を取得する方法を説明します。CredentialProfileStoreChainクラスを作成し、その認証情報を使用してAmazons3clientオブジェクト。資格情報は、SDK ストアまたは共有のAWS認証情報ファイルをデフォルトの場所に配置します。

この例では、Amazonランタイム.awscredentialsクラス。

```
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 ストアまたは共有のAWS認証情報ファイル

この例では、CredentialProfileProfileStoreChain クラスを使用して、プロファイルを取得する方法を説明します。資格情報は、SDK ストアまたは共有のAWS認証情報ファイルをデフォルトの場所に配置します。

この例では、クレデンシャルプロファイルクラス。

```
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認証情報を取得する

この例では、CredentialProfileProfileStoreChain クラスを使用して認証情報を取得する方法を説明します。資格情報を使用するファイルから取得します。AWS認証情報ファイル形式で保存されますが、別の場所に配置されます。

この例では、Amazonランタイム.awscredentialsクラス。

```
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
}
```

クラスの例共有資格情報ファイルとAWSCREDentials ファクトリ

SharedCredentialsFile クラスを使用して AmazonS3Client を作成します。

この例では、共有のAWS認証情報ファイル、作成AWS資格情報をプロファイルから削除し、その資格情報を使用してAmazons3clientオブジェクト。この例では、SharedCredentialsFileクラス。

この例では、クレデンシャルプロファイルクラス、およびAmazonランタイム.awscredentialsクラス。

```
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ことを除いて、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

使用している、AWS SDK for .NETおよび.NET Standard 2.0を使用するには、アプリケーションがAWS SDK for .NETNuGet を使用するのではなく、アセンブリ (DLLファイル) を直接呼び出すことができます。この要件を考慮すると、以下を実行する必要がある重要なアクションです。

- あなたは、取得する必要がありますAWS SDK for .NETアセンブリを作成し、プロジェクトに適用します。これを行う方法については、」ZIP ファイルをダウンロードして抽出します。 (p. 216)トピックAWSSDK アセンブリを入手する (p. 216)。

- 次のDLLをUnityプロジェクトに含める必要があります。AWSSDK.Coreと他のAWS使用しているService バージョン 3.5.109 以降AWS SDK for .NETの場合、.NET 標準 ZIP ファイルには、これらの追加の DLL が含まれています。
  - Microsoft.BCL.AsyncInterfaces.dll
  - System.Runtime.CompilerServices.Unsafe.dll
  - System.Threading.Tasks.Extension

- 使用している SCPPを使用してUnityプロジェクトをビルドするには、`link.xml`ファイルをAsset フォルダに追加して、コードの削除を防止します。`-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、各アセンブリに`preserve="all"`属性。以下のスニペットは、このファイルの例を示しています。

```
<linker>
    <assembly fullname="AWSSDK.Core" preserve="all"/>
    <assembly fullname="AWSSDK.DynamoDBv2" preserve="all"/>
    <assembly fullname="AWSSDK.Lambda" preserve="all"/>
</linker>
```

Note

この要件に関連する興味深い背景情報を読むには、http://aws.amazon.com/blogs/developer/referencing-the-aws-sdk-for-net-standard-2-0-from-unity-xamarin-or-uwp/。

これらの特別な考慮事項に加えて、バージョン 3.5 の変更点 (p. 65)Unity アプリケーションをバージョン 3.5 のに移行する方法については、AWS SDK for .NET。

Xamarin 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。

また、についての情報を参照してください。ポータブルクラスライブラリとXamarin (p. 63)。

の API リファレンスAWS SDK for .NET

-AWS SDK for .NETへのアクセスに使用するAPIを提供しますAWSのサービス。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 .NETAPI リファレンス。

上記の一般的な参考文献に加えて、以下の各例はの使用AWSのサービス (p. 69)セクションには、その例で使用されている特定のメソッドとクラスへの参照が含まれています。

ドキュメント履歴

次の表は、の前回リリースからの重要な変更点をまとめるものです。AWS SDK for .NETデベロッパーガイド。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。

更新履歴の変更	更新 - 履歴 - 記述	更新 - 履歴 - 日付
AWS SDK for .NET バージョン 3 ガイド統合 (p. 222)	2 つのAWS SDK for .NETバージョン 3 の開発者ガイド、「V3" と「最新」は、「v3" URL の下に 1 つのガイドに統合されました。	2021 年 8 月 18 日
設定ファイルの操作 (p. 200)	JSON 設定ファイルに関する情報が追加されたAWS.NET 配置ツール	2021 年 7 月 26 日
.NET Standard 1.3 からの移行 (p. 67)	.NET Standard 1.3 Support は AWS SDK for .NET人生のその終わりに来ています。	2021 年 3 月 25 日
AWS .NET 配置ツール (プレビュー) (p. 190)	のプレビュー情報を追加しましたAWS.NET 配布ツール。.NET CLI からアプリケーションを配布するために使用できます。	2021 年 3 月 15 日
バージョン 3.5 の AWS SDK for .NET (p. 65)	バージョン 3.5 のAWS SDK for .NETが解放されました。	2020 年 8 月 25 日
ページネーター (p. 52)	多くのサービスクライアントにページネータを追加しました。これにより、API結果のページネーション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 (p. 50)	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 (p. 211)	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を追加しました。	2020 年 8 月 7 日
S3 暗号化に KMS キーを使用する (p. 157)	S3 暗号化クライアントのバージョン 2 を使用するように例を更新しました。	2020 年 8 月 6 日
.NET Standard 1.3 からの移行 (p. 67)	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 (p. 5)	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 (p. 209)	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